ASM • Graphics • C • Pascal • BASIC • Database • CLint

The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal™

JULY/AUGUST 1989
Volume 7.4
$4.95 Canada $5.95

URBO CAL 5.5

ESENTATION MANAGER

C MUNICATIONS

F ANSFER
P COLS

A ERMINAL
P AM

T
P
5.5

- Graphics Timing
- ASM Coding
- Lint Utilities
- A Timer for BASIC

#21609*** PJS5832 MARAPR90
RON K UNZ
WALL STREET ANALYTICS INC
33-41 85TH ST
JACKSON HEIGHTS NY 11372

MILAM

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now it's easy to make your programs millions of times smarter.

More and more, programmers and work station builders are using DESQview as a development tool. The reason is simple. They can create powerful, multitasking solutions today for the millions of DOS PCs in use today. Solutions comparable to those promised for tomorrow by OS/2.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get access to the powerful capabilities built into DESQview —multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, mousing, data transfer, menu-building and context sensitive help.

Bells and Whistles

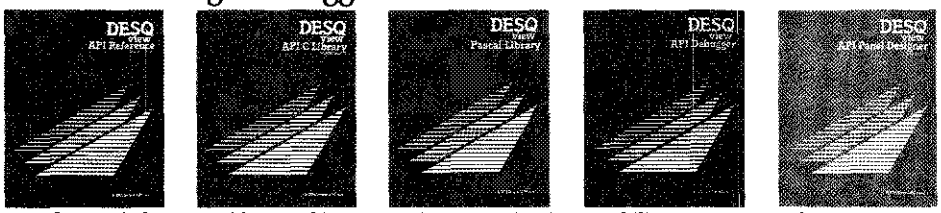
A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

Some of the applications under development right now using DESQview API Tools: CAD, medical systems, insurance, 3270 mainframe communications, network management, real estate, typesetting, point of sale, education, commodity trading, stock trading and online voting.

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make this possible: DOS Extenders from PharLap Software and AI Architects and DESQview support of these DOS extenders.

A Complete Interactive Toolkit —Including a Debugger


DESQ view API Reference
DESQ view API C Library
DESQ view Pascal Library
DESQ view API Debugger
DESQ view API Panel Designer

Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

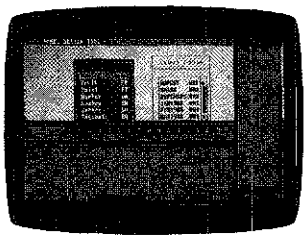
API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API C Library Here are C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Microsoft® C, and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Pascal Library The Pascal library provides interfaces for the entire set of API functions. It supports Turbo Pascal V4.0 and V5.0 compilers. Included are the API Reference Manual, source code for the library, and example programs.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be specified to report only calls of interest.

API Panel Designer This interactive tool helps you design windows, menus, help screens, error messages, and forms. It includes an editor that lets you construct an image of your panel using simple commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel, such as its position, size, and title. Finally, you can specify the locations and types of fields in the panel.



The Panel Designer automatically generates all the DESQview API data streams necessary to display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

More Tools are Coming Quarterdeck is committed to adding tools as needed by our users. To that end we have been working with Ashton Tate and Buzzwords International on dBASE III and dBASE IV translators. And in the works, we have BASIC and DOS Extender libraries.

If you'd like hands-on experience with the API Toolkit, write for information on our next API Conference, coming this August.

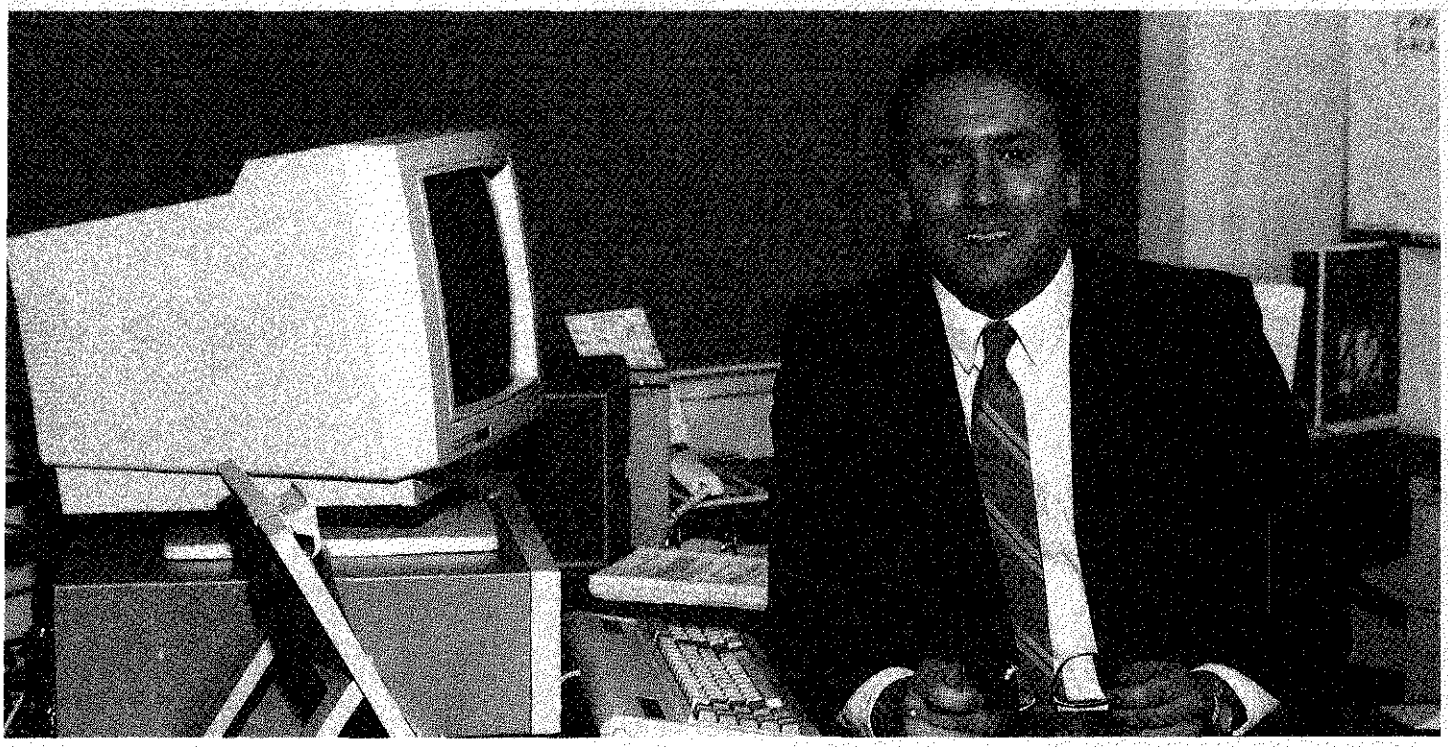
Bring New Power to DOS with the DESQview API Toolkit.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Number 147 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



When it comes to selecting compilers, MetaWare is the right choice!

Developing the next generation of software products is serious business. You need the best tools to produce the best code. You need **MetaWare**.

Superior Compilers

A compiler that handles large programs with ease, while producing the expected results, is the key to developing the highest quality applications. Many of **MetaWare's** customers say that **High C** ™ and **Professional Pascal** ™ are the highest quality compilers in the industry. They are reliable, and well documented, and their superior diagnostic messages help to produce better products more quickly. **There are no surprises with MetaWare compilers.**

Compiler Features

- ANSI Standard with extensions
- Generates small, fast executables
- Support of 80x87, Weitek 1167/3167, 68881, and Am29027 math co-processors
- Global common subexpression elimination
- Live / dead analysis
- Constant propagation, copy propagation
- Tail merging (cross jumping)

Multiple Platform Support

To support multiple platforms effectively you need the same compiler on each, eliminating the time and cost of developing and maintaining a new version of the product for each platform. **MetaWare** is *the only* company that provides the same compilers on a wide variety of platforms and operating systems, to both OEMs and End Users.

MetaWare uses its own **Translator Writing System** to create all of its compiler products. Common components and library functions are shared across the product line. Improvements to compilers are immediately realized on all platforms.

Supported Platforms

- 386: Extended-DOS/UNIX/XENIX
- PC: DOS/OS2
- Sun-3, Sun 386*i* , Sun-4,
- IBM: AIX/386, AIX/RT, AIX/370
- DEC VAX: VMS (host for cross compilers and TWS, not native)
- DEC VAX: Ultrix
- AMD 29K (Native/Cross)
- Intel i860 (Native/Cross)

MetaWare's Commitment

MetaWare's unique approach to business is its ability *and willingness* to support both OEM and End Users alike. The commitment to providing quality products doesn't end when you receive your compiler. Technical phone support is provided by engineers at a level of expertise consistent with the quality of **MetaWare's** products.

So when it comes to selecting a compiler company, **you need the best!**

Can you really afford anything less?

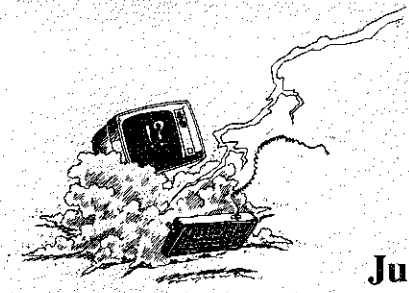
MetaWare™
INCORPORATED

Compiler Products for
Professional Software Developers

2161 Delaware Avenue
Santa Cruz, CA 95060-5706
Phone: (408) 429-META (6382)
FAX: (408) 429-WARE (9273)

MetaWare, High C and Professional Pascal are trademarks of MetaWare Incorporated; UNIX is a trademark of AT&T; Sun-3, Sun-4, and 386*i* are trademarks of Sun Microsystems, Inc.; PC-DOS, RT PC, and IBM are trademarks of International Business Machines Corporation; 386 and i860 are trademarks of Intel Corporation; DEC, VAX, Ultrix, and VMS are trademarks of Digital Equipment Corporation; XENIX is a trademark of Microsoft Corporation; 1167 and 3167 are trademarks of Weitek Corporation; 29K and 29027 are trademarks of Advanced Micro Devices, Inc; 68881 is a trademark of Motorola Corporation.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal™

Volume 7.4
July/August 1989

Contents

4 Notes from the Editor

6 Letters
Roger Knights, Larry Smith, P. E. Cogar, Arne Rohde, Robert Dunlop, and George Dinwiddie

12 Redirections: Pink Noise *by Frank D. Greco*
The latest news and views on the future-technology sector

20 On Graphics: Measuring Performance *by Michael Abrash*
A high-resolution timer for testing your graphics code

32 Communications: A Simple Terminal Program in Presentation Manager *by William S. Hall*
Accessing the OS/2 communications interface to transform your powerful OS/2 machine into a dumb terminal

44 Communications: PC Communications and Protocols *by M. Steven Baker*
An overview of PC serial communications hardware, protocols, and the new breed of fast modems

54 ASM: A Line of Code *by John Otken*
Better coding conventions for assembly language on the PC

61 ASM: Fast String Searching *by Adrian B. Crum*
Implementing the Boyer–Moore search algorithm in assembly language

64 Database: Database Normalization *by Bill Madison*
The theory and practice behind normalizing databases for performance

72 Pascal: The Future Arrives *by Tom Swan*
A preliminary look at Borland's enhanced Turbo Pascal 5.5 development tools that now support **OBJECT**s

75 C: Lint Utilities *by Rex Jaeschke*
A look at Lint utilities available for the PC

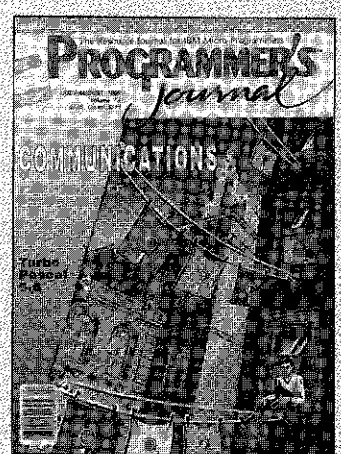
82 BASIC: A Millisecond Timer for BASIC *by Ethan Winer*
How to generate accurate timing delays across PC hardware in BASIC

86 BASIC: Exit Codes for QuickBASIC *by Murray L. Lesser*
An assembly language routine to add the ability to pass exit codes from QuickBASIC

90 Miscellaneous *by M. Steven Baker*
Communications
RamNet (Software Concepts Design)
DOS Extenders
DOS/16M (Rational Systems)
Graphics and Animation
PCX *Technical Reference Manual* (ZSoft)
PCX Programmer's Toolkit (Genus)
GRASP 3.1 (Paul Mace Software)
File Managers
AXOS (McSoft Incorporated)
Books
Microsoft Mouse Programmer's Reference

95 *PJ* Billboard

96 Advertiser Index



Communications used to mean shouting out the window or pounding on the ceiling with a broom. The modern age has made communications slightly more complicated, but *PJ* can help keep you from getting your wires crossed.

Artist:
Larry Milam

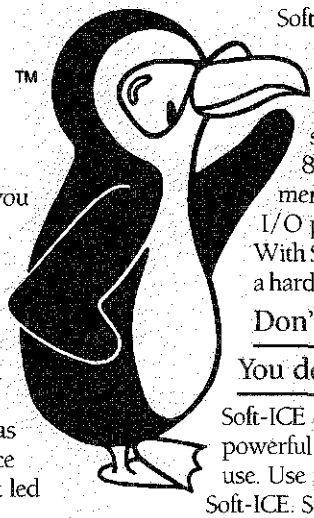
Art Director:
Ralph Bentley

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY. A debugging tool tough enough to handle the DOS Nasties.

New Version 2.0

Soft-ICE™



Nasty over-write? No sweat!

Soft-ICE memory range break points help you track down memory over-write problems whether you are doing the over-writing or another program is over-writing you.

Hung program? No problem!

When the system hangs, you now have hope. With Soft-ICE you can break out of hung programs no matter how bad the system has been trashed. And with Soft-ICE's back trace ranges you can re-play the instructions that led up to the crash.

Program too large? Not with Soft-ICE!

Soft-ICE runs entirely in extended memory. This means you can debug even the largest DOS programs. And since your program runs at the same address whether Soft-ICE is loaded or not you can find those subtle bugs that change when the starting address of your code changes.

System debugging? Soft-ICE is a natural!

Soft-ICE is ideal for full source level debugging of TSRs, interrupt service routines, self booting programs, DOS loadable device drivers, real-time kernels, non-DOS O/Ss and ROMs. Soft-ICE can even debug within DOS & BIOS.

How Soft-ICE Works

Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses the 80386 to provide real-time break points on memory locations, memory ranges, execution, I/O ports, hardware & software interrupts. With Soft-ICE you get all the speed and power of a hardware-assisted debugger at a software price.

Don't want to switch debuggers?

You don't have to!

Soft-ICE can run stand-alone or it can add its powerful break points to the debugger you already use. Use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated automatically.

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

"These may be the only two products I've seen in the last two or three years that exceeded my wildest expectations for power, compatibility and ease-of-use."

—Paul Mace
Paul Mace Software

Soft-ICE	$386
MagicCV	$199
MagicCV for Windows	$199
Buy Soft-ICE & MagicCV(W)	—Save $86.
Buy MagicCV and MagicCVW	—Save $100.
Buy All 3	—Save $186.

30 day money-back guarantee
Visa, MasterCard and AmEx accepted

1988 BYTE AWARD OF DISTINCTION

New Soft-ICE 2.0 features

- Back Trace Ranges
- Symbolic & Source level debugging
- EMS 4.0 support with special EMS debugging commands
- Windowed user interface

Nu-Mega™ TECHNOLOGIES

CALL TODAY (603) 888-2386
or FAX (603) 888-2465

RUN CODEVIEW IN 8K MagicCV™

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView in less than 8K of conventional memory on your 80386 PC.

NEW—Version 2.0 includes EMS 4.0 driver
Attention Windows Developers!
Version available for CVW.

P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607

Number **160** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers
PROGRAMMER'S Journal™

NOTES from the Editor

Editor
M. Steven Baker
Managing Editor
Robert E. Wanetick
Contributing Editors
Michael Abrash, Frank D. Greco
Tom Swan

Contributing Writers
Michael Abrash
M. Steven Baker
Adrian B. Crum
Frank Greco
William S. Hall
Rex Jaeschke
Murray L. Lesser
Bill Madison
John Otken
Tom Swan
Ethan Winer

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Production
Ralph Bentley
Scott E. Pringle
Illustration
Jim Carpenter
Advertising
Robert Tappan
Andrea McHone
Subscriptions
Gary Hayer
Administrative Assistant
Lori Fanslow
Founder
Greg Estes

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
503-747-0800
Fax: 503-746-0071

Subscriptions: 1 year (6 issues) $24
($30 in Canada and $40 elsewhere)
Change of Address
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Code and listings in *Programmer's Journal* appear "as is" and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Andrea McHone at 503-747-0800, or write to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

Programmer's Journal is produced using the following software products: Ventura Publisher, Aldus PC PageMaker, and CorelDraw. The PostScript files are output on a Linotronic 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company. Second class postage paid at Springfield, OR, and additional mailing offices.

POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

© 1989 OAKLEY PUBLISHING COMPANY

In the fast-paced PC world, the meaning of "communications" appears to change with each product generation. New hardware developments continue to push the limits of technology and cost—evidence the fast (by yesterday's standards) 2400-baud modems being displaced by faster modems (9600 baud and above) available at ever more reasonable prices. We chose to focus on communications in this issue and hold our networking material for later. There was just too much material on networking to share editorial space with PC communications. The DOS networking market is also in flux, with NetWare 386, OS/2 Lan Manager, SUN NFS, AT&T RFS, OSI, and TCP/IP, and so we'll postpone our coverage to a future issue.

The Object-Oriented Language Wars Begin

Borland and Microsoft have both just released new versions of Pascal that support objects. Borland added object-oriented extensions to Turbo Pascal based on Apple's Object Pascal and ideas from C++. Borland's Turbo Debugger was also enhanced to support objects. Microsoft developed their new QuickPascal from a French product modeled after Macintosh Object Pascal. Microsoft Pascal supports most of the Borland syntax. These products are also the prelude for the C++ language translators being developed at both companies. Object-oriented programming tools may be needed by developers, but they are also necessary for vendors to add value to their products.

DOS Extenders Revisited

DOS operating system software (OS/2) continues to lag behind CPU developments. Two years ago, I started exploring the 386 DOS extenders. A question for me at the time was: How long would there be a market for these products? Although both 286 and 386 DOS extenders have been available for some time, a bit of a watershed has been passed. Lotus has chosen to use Rational Systems *DOS/16M* DOS extender for Lotus 1-2-3 Release 3. Prior to the Lotus announcement, a number of products had been marketed using DOS extenders—Borland's Paradox-386 using PharLap, CADKEY using A.I. Architects, FoxBASE 386 using IGC, and so on. But many of these products are targeted for specialty or niche markets in CAD/CAM, scientific computing, and program development. Microsoft even previewed a 386 DOS extender for Windows at their Systems Forum this spring.

With Lotus selecting Rational's *DOS/16M*, the DOS extender products have been fully legitimized. Lotus continues to have the number-one-selling PC application. Remember that Lotus selected the EMS expanded memory interface and thus forever legitimized EMS in the PC world. Lotus has also signed on to the Virtual Control Program Interface (VCPI), along with the original sponsors of VCPI—PharLap, Quarterdeck, A.I. Architects, Qualitas, Quadram, and Rational Systems. VCPI (jointly developed by PharLap and Quarterdeck) is a software protocol designed to avoid conflicts between protected mode software from different companies. Although not an official sponsor, Compaq has also incorporated support for VCPI in their latest release of their CEMM memory manager. The VCPI specification is available at no cost from PharLap Software (60 Aberdeen Ave., Cambridge, MA 02138). The PC market now sports both mainstream spreadsheet and database systems using DOS extenders. Which other memory- or power-hungry applications are next?

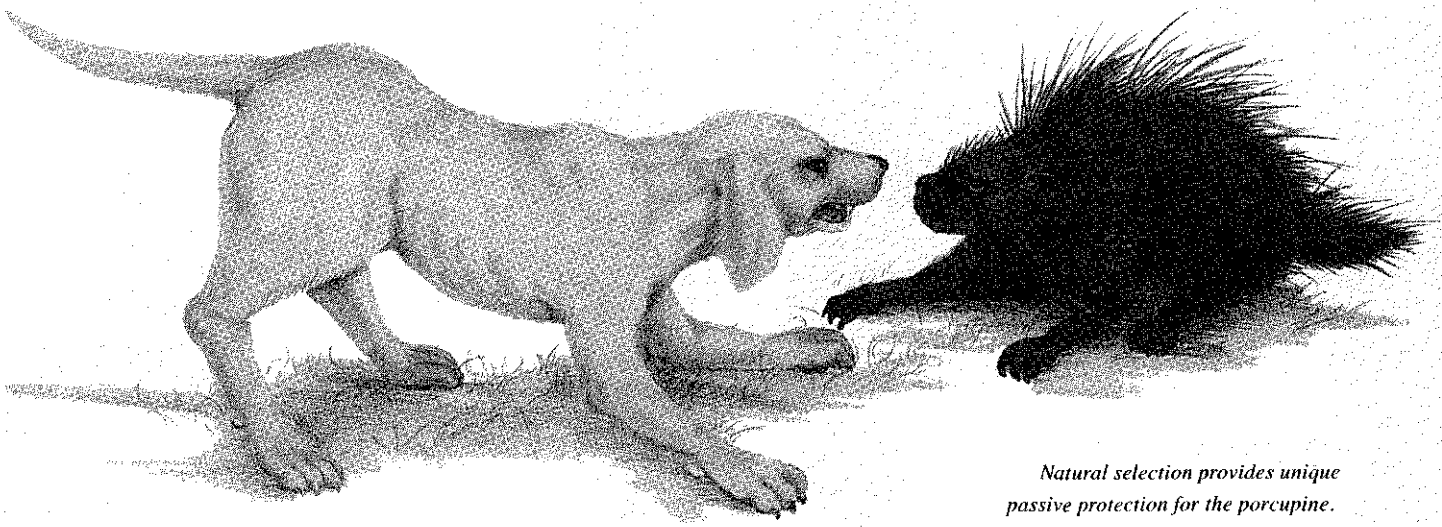
Ah . . . but for a Bit of Praise

We all occasionally need a pat on the back for a job well done. *Programmer's Journal* was recently selected by the Computer Press Association as runner-up for the best computer magazine with a circulation under 50,000. The winner in our category was *VARBusiness*, a controlled-circulation magazine with well-written stories and four-color graphics throughout. *PC Magazine* won the above-50,000 category for the second year in a row. *PJ* was judged as follows:

"Here is living proof that excellent writing and editing can make what could be humdrum sparkle. *Programmer's Journal* can't boast spiffy graphics, let alone photos. But its sprightly, literate and unassumingly personal style make it more appealing to read than any other entry in this category. Better yet, it is full of practical information and guidance presented with superb clarity and perspective. Never let anyone say that a computer magazine can't have personality and charm . . . along with lines of code."

We appreciate the compliments from our peers and aim to maintain our personality in this and coming issues. —*Steve*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Natural selection provides unique passive protection for the porcupine.

The Activator - Natural Selection For Software Protection

Inventor and entrepreneur Dick Erett explains how "The Activator" provides sane protection for your intellectual property.

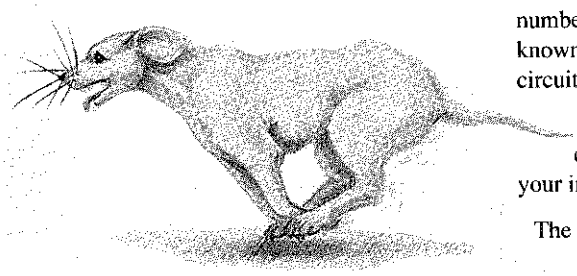
"In any industry, just as in nature, the process of natural selection raises one solution above another. Natural selection is the most elegant of engineers.

In the area of software protection The Block has been selected by the marketplace as the solution that works. Over 500,000 packages are protected by our device.

For the past 4 years our philosophy has been; *'You have the right and obligation to protect your intellectual property.'*

A New Ethic For Software Protection

In allowing end-users unlimited copies of a software package and uninhibited hard disk and LAN operation, The Block has created a new ethic for software protection.



By removing protection from the magnetic media we remove the constraints that have plagued legitimate users.

They simply attach our key to the parallel port and forget it. It is totally transparent, but the software will not run without it.

A New Technology For Software Protection

Our newest model, The Activator, builds on our current patented design, and establishes an unprecedented class of software protection.

We have migrated and enhanced the circuitry of The Block to an ASIC (Application-Specific Integrated Circuit) imbedded in The Activator.

This greatly improves speed and performance, while reducing overall size. Data protection can also be provided.

Programmable Option

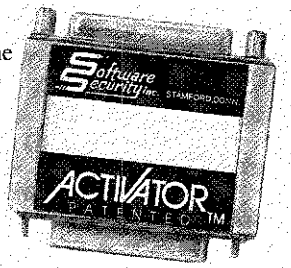
The Activator allows the software **developer** the option to program serial numbers, versions, or other pertinent data known only to the developer, into the circuit, and access it from the program.

Once you program your part of the chip, even we have no way to access your information.

The ASIC makes emulation of the device virtually impossible. It also presents an astronomical number of access combinations.

Full 100% Disclosure

Since The Activator is protected by our patent we fully disclose how it works. Once you understand it, endless methods of protection become evident.



Just as no two snowflakes are the same, no two implementations of The Activator are identical. And like the snowflake the simplicity of The Activator is its greatest beauty.

We never cramp your programming style or ingenuity. Make it as simple or complicated as you desire.

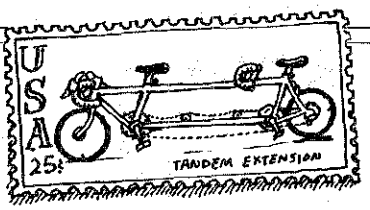
Let us help safeguard what's rightfully yours. Please call today for additional information or a demo unit. *It's only natural to protect your software.*"

1-800-333-0407 ext.103
In Connecticut 203-329-8870
Fax 203-329-7428

Software Security inc.
1011 High Ridge Road
Stamford, CT 06905

Unlimited Copies • Programmable • No Batteries • Small Size • Fast • Patented • Data Protection

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



COBOL Comments

I appreciate the coverage you've given to COBOL-85 in *PJ* 7.1, but the article "COBOL Comes of Age" contains some errors:

1. On page 85, first column, the first coded example should terminate in a period, not an END-IF. There is no END-IF in COBOL-74; using one has muddled the point the author was trying to make.

2. On page 86, first column, the last coded example shows a conditional variable being used as the "selection subject" in an EVALUATE verb and its associated condition-names being used in the WHEN phrases beneath it. This is not allowed in COBOL-85, as a close reading of its rules reveals. Tandem's COBOL, which the author was using, allows it as an extension; no other vendor does.

Although coding this way is natural and *should* be allowed, it's doubtful that the language will be officially extended. The CODASYL COBOL Committee in February considered my proposal to do so but couldn't agree on what to do in the case where the variable is a table element: Should a subscript be required after the variable, the condition-name, or both? Rather than just prohibit the use of table elements, they threw the baby out with the bath water and rejected the whole proposal.

3. On page 86, second column, the paragraph after the last coded example should have the following boldfaced words inserted to make its point clear: "This example also illustrates a practical use of the in-line PERFORM ... END-PERFORM **WITH TEST BEFORE** construct: table handling."

4. On page 89, first column, the first coded example shows the BY CONTENT phrase in the Procedure Division header (never allowed) of the called program, when it should be in the CALL verb of the calling program. And the second paragraph after the example says that BY CONTENT means that the called program "may not" modify the corresponding parameters in the calling program by changing its own parameters. Those words should be "cannot"; BY CONTENT isn't something the programmer of the called routine is concerned with.

5. This same coded example uses an unnecessary INITIAL clause in the PROGRAM-ID paragraph in the contained program; it's unnecessary since it has no WORKING-STORAGE Section to have its items initialized when it is called. More important, by not showing a W-S Section, the example fails to illustrate some of the most significant features of contained programs: their ability to define their own variables (like "procedures" in Pascal and PL/I), and to modify items in the calling program that weren't even passed to them as parameters, provided the items were defined with the GLOBAL attribute. (Incidentally, contained programs in effect provide one of the new features the author calls for at the end, "a PERFORM statement that allows parameter passing.")

Another important data attribute is EXTERNAL, which facilitates the testing of subprograms separately, first using a simple calling program to pass in test data and files that are both external and global, then having the main production program make the call, without having to establish and disentangle a complicated "test harness." This provides (some of) another future feature the author desires, "greater subprogram independence."

6. On page 89, second column, under "String Manipulation," the lead-in to the last example states, "When this feature [reference modification] is used in conjunction with the STRING and UNSTRING commands . . . , we are able to do some powerful manipulation of string data," but the example fails to show any use of reference modification. Here is an example that does:

```
STRING DATA-1 (1:3), DATA-3 (4:3)
INTO DATA-4
```

resulting in

```
ABCPQR
```

7. On page 89, second column, under "New Level-88 Capabilities," the use of a FALSE phrase in an 88-level entry is shown, along with a SET <88-level-entry> TO FALSE verb. This is not part of the COBOL-85 standard, although it is in the CODASYL *Journal of Development* and will be part of COBOL-95 (presumably). For now, it is only a Tandem extension.

Roger Knights
SPC Systems USA
Seattle, Washington

GRAPHICS!

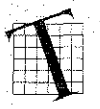
Add lightning fast graphics to your programs quickly and easily through the popular PCX file format. Why reinvent the wheel? Make your programs immediately compatible with hundreds of packages from Aldus PageMaker to ZSoft's PC Paintbrush with these linkable graphic libraries.

"A Professional Class Product" - DBAdvisor, Jan '89

NEW! Version 3.5 of the **PCX Programmer's Toolkit** gives you over 60 powerful functions to manipulate bitmapped graphics. Use Virtual screens, Super VGA modes, LIM 4.0 support, a 300 page manual, 9 utilities including screen capture and display, and the fastest routines on the market. **$195**

Need Special Effects, but caught in a GRASP?

Why create a demo when you can create the real thing? Don't be trapped in a slideshow editor or demo program when you can use **PCX Effects** and your favorite programming language. A complete Music Language and spectacular special effects for exploding your graphics! **$99**



Blazing Graphics Text

With **PCX Text** you can display text with graphics as fast as it always should have been. Display characters, strings, input fields, font scaling, background transparency, and more. Includes a font editor, fonts, and conversion utility for blazing graphics bitmapped text. **$99**

All packages support 12 compilers for C, Pascal, Basic, Fortran, Assembly, and Clipper. All modes of the Hercules, CGA, EGA, VGA, and Super VGA adapters are supported, up through 800x600x256 (22 modes in all). Assembly Language source code is optionally available. Trademarks are property of their respective holders.

No Royalties! 30-day Money Back Guarantee.
VISA/MC/AMEX/COD/PO accepted.

G·E·N·U·S
MICROPROGRAMMING
11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737

1-800-227-0918

Number **261** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Accruing Interest

I read with a certain amount of interest M. Steven Baker's column "386 Now" in *PJ* 7.2. About a year ago, I purchased a Hauppauge 386 main board for my aging PC. Like him, one of the first things I did with my 386 was to rewrite all my string move routines to deal with double words instead or words (or bytes), but the 386 is a honey of a chip and has quite a few very interesting new instructions that make programming easier and fun. Most of what I have done to date has been to rewrite small portions of code to take advantage of these new instructions. I have not yet written a strictly 386 program. If you would be interested, I would be glad to send *PJ* some examples of what I have done.

I also read with some interest the letter exchange on Jake Richter's article in *PJ* 6.6 concerning memory above 1 megabyte. When I taught assembler programming several years ago, I became aware of the routine **GATE_A20** called by the IBM POST on the AT. A more hardware-minded colleague of mine also pointed out the funny gating line on the main board schematic. The AT/BIOS listing is not well indexed; so it took some time to track down the routine in question. To say the least, I found IBM had carried compatibility to new heights of insanity, and the clone manufacturers followed suit blindly.

Having an 80386 to play with, I began experimenting a little. I have since written a small TSR kernel (DOSEDIT and printer interface) that runs in this "new-found" memory and a printer spooler that keeps its 48Kb buffer there. There are all kinds of complications in doing so (as Richter rightly points out in his letter), but the gain in normal DOS memory is rewarding. If *PJ* would be interested, I would be glad to send a small article plus source code to you.

I continue to enjoy *PJ*. Keep the ball rolling.

Larry Smith
Friedland, West Germany

[Thanks for the feedback and the kind words. We appreciate your interest and look forward to seeing your articles. —REW]

People are talking about us.

When professional FORTRAN programmers develop or port large programs they use Lahey's F77L-EM/32 and F77L-EM/16, *PC Magazine's 1988 Technical Excellence Award Winners.* F77L-EM/32 is a fast 32-bit protected-mode compiler that accesses up to 4 gigabytes of memory on 80386s. F77L-EM/16 gives 80286 users the power to create 15 megabyte programs. These protected-mode FORTRANS include the features that have made them, and our F77L and Lahey Personal FORTRAN, market leaders: full ANSI 77 Standard, VAX and IBM VS extensions, fast compilation, comprehensive diagnostics, and a powerful debugger.

TECHNICAL EXCELLENCE PC MAGAZINE WINNER COMPILERS/LANGUAGES F77L-EM/32 AND F77L-EM/16 LAHEY COMPUTER SYSTEMS, INC. 1988

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

Number **44** on the Reader Service Card

RADs™

works the way you think.

Rapid Application Development System

We take you from Concept to Prototype to Finished Program to Demo to Tutorial

Read our line scenario:

"So, Your	• Boss • Client • Market
says: "I need a:	• Program • Prototype • Demo • Tutorial • Presentation • All The Above
"with:	• Nested Menus • Pop-up windows • Pull-down Menus • Light Bars • Original User Interface • Music
"Oh, and in the prototype, demo, and tutorial could you make it:	• Interactive • Self-Running • Both
"And it needs:	• User Input • Simulated Report Generation • Active Screen Areas for User Input • Dialogue Boxes • Variable Math Calculation • GO SUB Routines and much more
"Oh, and one more thing, I need to:	• Make as many copies as I want with no license fees
"And:	• No screen in front promoting the product it was developed in.
"And:	• Free 800 # telephone support
"Oh, and just one more thing, I'd like to use the screens from the prototype in the program we're writing in:	• C • Basic • Pascal • dBase • Clipper
"And the routines should also do:	• Screen Handling • File Handling • Mouse Support • Serial Port Control

"And could I have that in 2 days please?" Sound familiar?

NO Problem, if you've got ***RADs !***

GENESIS

GENESIS DATA SYSTEMS
8415 Washington Place N.E.
Albuquerque, NM 87113 (505) 821-0425

COMPLETE SYSTEM - $299.95
Demo Available

To Order or For Product Information Call:

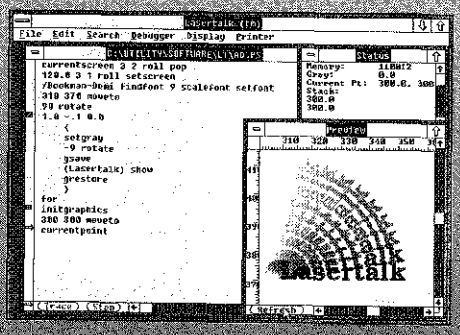
1-800-777-1437

Number **274** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Master PostScript®

Language Programming - On Your PC



Lasertalk PC

A PostScript language development environment which provides interactive access to the PostScript interpreter, a PostScript language debugger/editor, on screen previewer, dictionary browser and Adobe Systems' operator descriptions online. Requires PC/AT or greater and a PostScript printer. Includes Microsoft® Windows runtime. Suggested Retail Price $395. For more information, see your dealer or contact Emerald City Software at (415) 924-4040.

Number **241** on the Reader Service Card

We Could Try Carrier Pigeons

Please don't send me any more reminders about my subscription—I have already renewed for the next two years. Not that I mind receiving the reminders (it's better than stony silence and it's nice to feel wanted), but you are now wasting both your time and your money.

What I am actually writing about is the non-supply of your January/February 1989 issue. I received the March/April issue today, and so now I know the earlier issue is just not going to show up. It could have been sent by stagecoach and it would have reached me by now.

You don't hate Australians, do you? We won the America's Cup fairly on the water, and we lost it graciously on the water. Please don't confuse us with the New Zealanders, who like to fight their sporting events in the courts. Australia is a long way from the northern hemisphere, and we feel somewhat neglected when you cut off our supply of "brain food."

P. E. Cogar
Narrabeen, N.S.W., Australia

[Apologies for sending you so many reminders and for losing your issue. Your guess about the missing issue being sent by stagecoach was not too far off the mark. Our overseas shipper informed us in April that the January/February issues were delayed for a

PJ CODE

Programmer's Journal Code DiskScription—

Typing in long code listings uses up a lot of valuable time, and calling up bulletin boards doesn't appeal to everyone. So what does the modern programmer do? Get a ***PJ Code DiskScription***, of course!

What's a PJ Code DiskScription?
All the code listings from six consecutive issues of *Programmer's Journal*, sent directly to you every other month on disk, for the money-saving subscription price of only $39.95.*

You'll get all the upcoming installments of Michael Abrash's EGA/VGA ASM code. You'll get more of Tom Swan's handy Pascal code. You'll get great code from all ***PJ***'s authors, some of the country's hottest programmers.

You'll receive your code disks in the mail every other month throughout the year. Learn from ***PJ*** code or use in it your own programs. Each disk contains all of the program listings published in ***PJ*** plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***

If purchased separately, the regular price for these six disks is $60.00. **So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!**

*GUARANTEE: If you are dissatisfied with your **PJ Code DiskScription** for any reason, we will gladly send you a full refund on all unmailed diskettes.*

Code disks from six issues, only

$39.95*

***PJ CODE* DELIVERED DIRECTLY TO YOU—**
GET YOUR *PJ Code DiskScription* TODA

All *PJ Code DiskScriptions* must be prepaid.
Return the card on the opposite page with your payment.
For faster service with VISA/MC call toll-free: **800-234-0386**

* **Prepaid, US only.** Call or write for prices outside the US.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HARDCORE SOFTWARE

Stop fooling around. It's time to get hardcore about software. With Microsoft.

We'll give you all the resources you want. Tens of millions in R&D funding. Along with one of the most elementary tools for thinking — a door, which leads to your own private office. All backed by management that truly *does* speak your language, because they probably helped write it.

We're serious about software design. If you are too, then apply right now for one of these opportunities.

Software Design Engineers

We're working on everything from operating systems, windows, sophisticated graphics and networking, to compilers, powerful application software and more. In fact, we're working on some truly visionary ideas we can't even reveal yet. You could be too, if you have programming experience and a background that includes micros, "C", 8086, 68000, Windows, Macintosh® Toolbox, OS/2, MS-DOS, or UNIX™/XENIX.

Technical Leads

You can be one of the key members of our teams that develop leading edge software for Microsoft in our operating system, Windows, networking or application software groups.

We're looking for expert designers with technical lead experience to lead a team of experienced developers in the design and development of the product. You should have a BS or MS in CS and preferably 3 or more years of software development experience, with 1 or more years as a successful technical lead. Advanced knowledge of OS internals, networking, windowing environments, or application design and development is desired.

If you qualify, Microsoft offers you an opportunity to live and work where the quality of life is high and the cost of living is low — the beautiful Pacific Northwest. Along with amenities such as a health club membership, workout facilities and parcourse, plus an array of benefits.

To apply, please send your resume in confidence to: MICROSOFT CORPORATION, Recruiting, Dept. SVHCSW-07PJ, 16011 N.E. 36th Way, Box 97017, Redmond, WA 98073-9717.
We are an equal opportunity employer.
No phone calls, please.

Microsoft®

Number **232** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

month while the post office in Santo Domingo, Dominican Republic, was being remodeled. Please be assured that we don't hate Australians (although I am getting a little tired of Paul Hogan). —REW]

Yet Another Country Heard From

Thank you for getting your printer to send page 38 of Volume 7.1 by air mail. Now I have a month or two to read this page thoroughly before I get the remainder of the magazine. . . .

Ray Duncan had a FORTH apologia in Volume 6.6, but . . . it will take more than an apologia to convince me that FORTH is a viable alternative for developing commercial-grade software. I originally gave up on FORTH about eight years ago, after purchasing a FORTH system that could not exchange data with the manufacturer-supplied operating system on the same computer, would only support disk drives with a better specification than those supplied by the computer manufacturer, and would not support single- and double-density disks on the same system.

Since then, I have seen very few programs that were obviously written in FORTH, but a few hours spent searching for the default screen colors in Quaid Analyzer and Quaid Disk Explorer have convinced me that they were. The experience has also convinced me that machine-code debugging of FORTH programs (the only option when source code is not available) is considerably more difficult than for non-interpreted languages.

Lack of communication with customers . . . afflicts several . . . U.S. companies I have dealt with. One cross-compiler I purchased had a bug that would plant an apparently random byte value somewhere within the linked code, making it very difficult to rely on the results. Repeated communication with the company gave no results until I wrote to the president of the company, who had appeared in advertisements claiming that their software was used to control submarines, suggesting that this [claim] was not surprising if they supplied substandard software. This brought an immediate reply and a refund. . . .

Arne Rohde
Auckland, New Zealand

[We were quite concerned to read about the problems you have had with some of our advertisers. Rather than print your entire letter and assume that each company you named would respond to your concerns, we have asked our advertising manager to contact the companies. We deal with our readers with honesty and integrity, and we expect the same high standards from our advertisers. (Incidentally, only one of the advertisers you had complaints about appears in this issue . . . hmmmm.) —REW]

Use Clipper for Filters

First, let me say that I enjoyed your article on using filters to convert database files and that the thrust of the article (using portable C routines to create the filters) was not lost on me. However, I would like to point out that much of the work you did might have been avoided by selecting a different dBASE clone. I believe that many of the limitations you ran into might have been solved had you investigated further. Clipper has a limit of 64Kb per field, which would have handled the 6K to 8K fields from your other systems. In addition, with Clipper's ability to read DOS files directly, you might well have been able to write your filters in Clipper directly. Finally, I might point out that, based on your "barebones" description of the system requirements, the application could have been quickly coded in Clipper as well, with no run-time license fees at all. Again, let me say that I found the C code a useful resource and that it is going into my library of routines.

Robert Dunlop
Walnut Creek, California

[I appreciate your comments about using Clipper for development. I would have still been forced to live within the 4000-byte limits of dBASE for the database files. Although I developed a standard menu front-end, the other public agencies and persons receiving the database occasionally need access to the raw datafiles for custom queries or special purposes. Unfortunately, this compatibility would exclude use of Clipper's more liberal file limits. —MSB]

Add an Option

I found M. Steven Baker's "Converting Database Files Using Filters" (*PJ* 7.3) to be very interesting and potentially very useful. While I have not played with the code presented, I do have some comments on the implementation, in particular, of his "standard template for writing filters in C."

In the article he states, "If CUT is invoked without any arguments, detailed help for proper usage is printed to stderr." Unfortunately, this prevents the program from functioning as a filter in the UNIX sense because stdin is no longer available as input. A check of the code confirms my suspicions that at least one input file is required, in spite of the comment that appears above the function **dofiles()**. In my own, evolving "standard template," I have eliminated this problem by providing an option, *-?*, that prints a usage message. This generally works on other programs also, since *?* is rarely a valid option if not used for "help."

Owing to my quest for simpler, more flexible arrangements for parsing command lines, I was quite interested in the **getopt()** function. Unfortunately, no code was provided. Nor was there an explanation that would enable implementation of a functional duplicate. For the benefit of those of us without access to the AT&T or Borland versions, could you describe the interface and workings of **getopt()**?

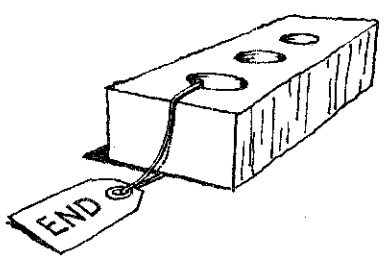
George Dinwiddie
Mitron Systems Corporation
Columbia, Maryland

[The code in CUT that handles command-line options will provide a usage message if **any** *bad option is given. I use this to give help if either -?, -h, or -H is given. This eliminates the need for an explicit help option. You are perfectly right about this version of CUT: At least one file must be provided on the command line. The AT&T UNIX CUT command will get input from stdin if no files are given on the command-line. This can be easily added to the code by setting the input file pointer to stdin. I might note that these programs were coded as quick tools.*

Under DOS, I hardly ever use pipes, since they are implemented simply as temporary files. Also, if a casual user invokes a program that expects input from stdin, they are provided with a system that looks like it's died. The user must know to enter a Ctrl-Z to close stdin. Also, the programmer can't control file buffering to speed operation using pipes. I have recompiled and relinked the file I/O libraries in both Turbo and Microsoft C using the library source code (available at additional cost) with 8K buffers so that these filter operate much faster. I've mostly used these tools on data files of 10 to 15 megabytes.

Two versions of **getopt** *source code are provided on the 7.3 issue disk (also available from the bulletin boards that carry PJ code). One version is the official AT&T version released to the public domain along with a manual page from USENET comp/sources/misc group. The second is a derived version of getopt from the GNU project's diff utility. —MSB]*

Programmer's Journal likes to hear from its readers. Please address correspondence to Managing Editor, Programmer's Journal, 150 N. 4th St., Springfield, OR 97477. You can also contact us via fax (503-746-0071) or MCI Mail (user name OAKLEY). All letters become the property of *Programmer's Journal* and may be edited for publication.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

S O F T W A R E S E C U R I T Y

WHETHER REPORT.

Whether you're a software developer writing new applications for the IBM or Mac, or a PC user securing proprietary data files, software and data protection has never had a brighter silver lining. For a number of very good reasons.

Beginning with the 'whether-expert' Rainbow Technologies. And ending with its Software Sentinel family of hardware keys. Starring five models that fit virtually any software program or data file you need to protect.

There's the best-selling SentinelPro for the IBM PC/XT/AT, PS/2 and compatibles, and even the Atari ST. Known worldwide for its virtually unbreakable security. And its ASIC technology. And its invisible operation. A close relation, the Sentinel-C stands at-the-ready for custom configurations and multiple software packages.

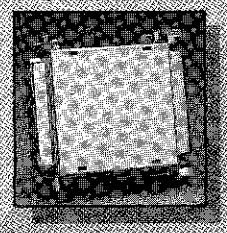
In the Apple market, security-minded Mac software developers turn to Eve. For completely transparent operation and world-class security of the protected software. Just by plugging Eve into the Mac ADB connector.

PC users wanting a low cost, user-friendly solution to the problem of securing sensitive data can call on the DataSentry. Using a proprietary Rainbow algorithm or DES, the DataSentry encrypts data files on individual PCs, protects modem transmissions and secures data on local area networks.

Rainbow's latest protection strategy is the SentinelShell—that lets users place a 'shell' around existing, off-the-shelf programs. Because access can be limited to those issued a key, libraries, universities and corporations can very simply guard their software investments.

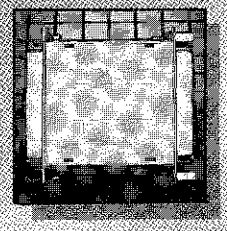
Whatever your whether, Rainbow Technologies has the software and data protection products that make the difference. For more information, call 714-261-0228 in the U.S., or contact Rainbow Technologies Ltd. in the United Kingdom for the distributor nearest you. Whethercasters are standing by.

SentinelPro™



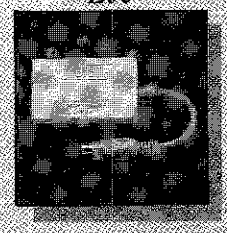
• Runs under DOS, OS/2 and Xenix • Algorithm technique (Never a fixed response) • External parallel port installation • Minimal implementation effort • Higher level language interfaces included • 100 times faster than fixed-response devices (1 ms) • ASIC design for reliability

Sentinel-C™



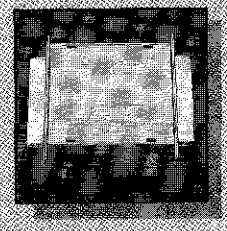
• Protects multiple packages with one device • 126 bytes of non-volatile memory programmed before shipment of the software • Rainbow supplies a unique adapter for programming the unit • Higher level language interfaces included • Runs under DOS, OS/2 and Xenix • External parallel port installation

Eve™



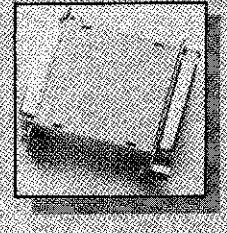
• For the Macintosh SE and II • Complies with Apple Desktop Bus Interface requirements • Rainbow-assigned developer passwords to prevent tampering by other developers or sophisticated "hackers" • 7 locks per key, usable individually or in combination, on one or up to seven applications

DataSentry™



• Completely user-installable • Pocket-sized external device • Menu driven, user-friendly interface • Single- or multi-user security system • Audit trail, log-on identifiers and automatic encryption/decryption of entire directories • Secures data transmitted by modems • Prevents recovery of data by utility programs

SentinelShell™



• Runs under DOS on IBM PCs and compatibles • Protects without requiring access to the source code • Completely transparent to the end user • User-friendly software • Pocket-size key attaches quickly to any standard PC parallel port • ASIC design for reliability

RAINBOW TECHNOLOGIES

18011-A Mitchell South, Irvine, CA 92714 • (714) 261-0228 • TELEX: 386078 • FAX: (714) 261-0260
Rainbow Technologies, Ltd., Shirley Lodge, 470 London Rd., Slough, Berkshire, SL3 8QY, U.K., Tel: 0753-41512, Fax: 0753-43610

© 1989 Rainbow Technologies. All product names are trademarks of their respective manufacturers.

Number **222** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pink Noise

by
Frank D. Greco

Well, I'm back.

After what seemed like an eternity away from this column, I'm ready to bring *PJ* readers some more valuable tidbits from the future technology sector. A short while ago, mother nature selected my house as a nice target for a lightning bolt and turned us on, so to speak. From the evidence, the bolt came through the electrical sockets that powered my computer equipment (lucky me). In a microsecond, I lost about $15,000 in computer equipment (not to mention two televisions, a CD player, a home security system, and the main circuit board in our furnace) to a megadose of current. It was a sobering experience to find two Hayes modems and a couple of answering machines half melted.

Luckily for me, I was not using my machine at the time. And if that wasn't enough, shortly after the lightning experience, I spent a brief stint in the hospital with some surgeons doing some cutting (but no pasting) correcting a minor problem. But all of that is finally over. Let's turn the burners on again.

Personal Computers, the Next Generation

What I think will be the start of the next generation in personal computing, the SPARC-Station 1, was finally announced by Sun Microsystems on April 13. This 12.5-MIPS machine (also known by its code name, "campus") will redefine what we mean by "personal" computing. Priced competitively around $12,000, this machine will give Steve Jobs's NeXT machine a run for his money. The SPARC-Station 1 includes a 20-MHz Fujitsu SPARC chip yielding 12.5-integer MIPS (normalized to VAX MIPS, of course); a Weitek math coprocessor; 8 megabytes of RAM; two serial ports; an Ethernet connection; a 19-inch, high-resolution, black-and-white display; two 104-megabyte internal SCSI hard disk drives; a 1.44-megabyte, 3.5-inch, DOS-compatible floppy disk drive; an optical mouse; SunOS with all software development tools; a windowing system (X/NeWS will be available in July for a nominal upgrade); sound capability; and software emulation of MS-DOS. Wow!

Power and Portability

This machine will initiate a new direction in small computers. Providing complete binary compatibility with any other computer manufacturer that bases its machine on the SPARC architecture, this relatively low-cost machine will spur the SPARC industry. Some smaller software development houses will now be able to afford a powerhouse machine; more customers will view the SPARC-Station 1 as economically viable; and software vendors will be able to lower their prices, thereby increasing demand for their products. Apparently, one anonymous customer put in an order for 20,000 workstations immediately after the announcement.

The affordability of the low-end SPARC-Station (bona fide developers can get a forty-percent discount through Sun's Catalyst program) will spur small houses to develop software for this machine. Although some may consider this machine expensive, it is a 12.5-MIPS machine, and, given the upcoming SPARC competition, a logical extrapolation would price it around $5,000 in a couple of years. A Sun engineer told me that the entire motherboard costs Sun only $800 in parts. That's less than the Intel 486 chip alone!

Besides the low-end SPARC, the Sun crew announced higher-performance SPARC-based machines, a 68030-based machine, and graphics accelerators. In addition, I heard that the Sunview API and X/NeWS window systems were already ported to the graphics accelerators and that window performance was substantially "snappier."

I Font You to Font Me

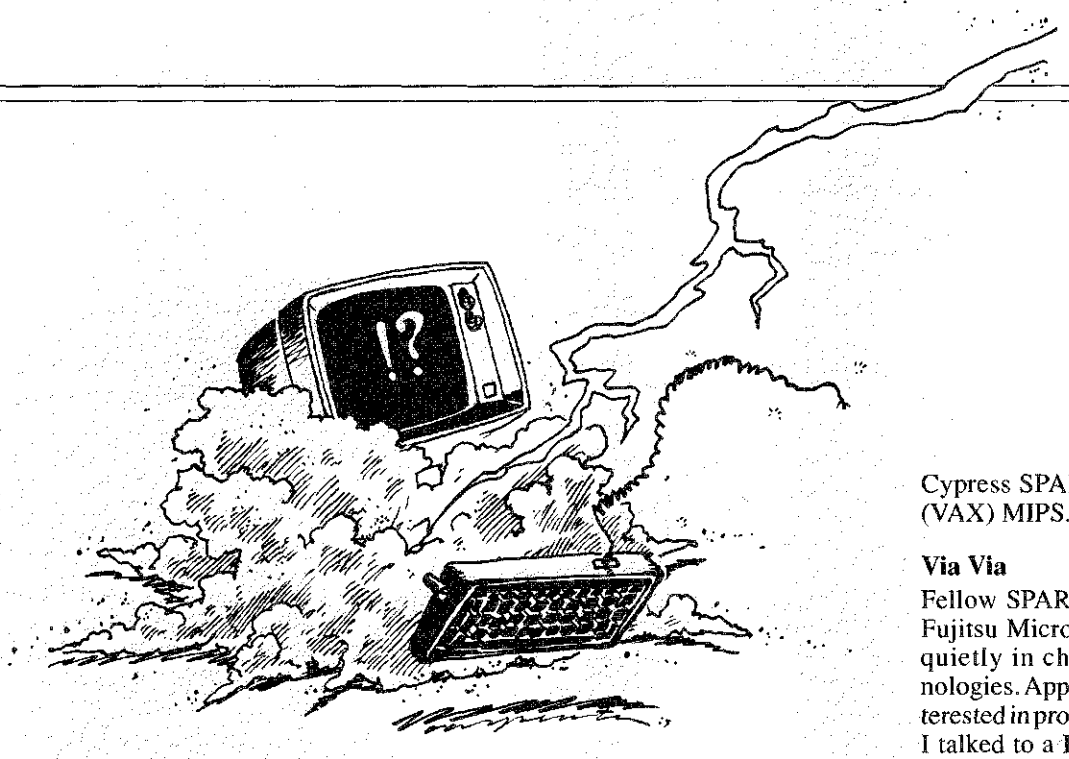
If you recall, I had mentioned that the quiet acquisition of Folio by Sun was extremely important to the Mountain View denizens. Folio's font technology facilitates the creation of outline fonts from bit-map fonts. As I surmised, this technology has been successfully merged in the X/NeWS window system. And since Sun has direct font licenses with the major font foundries (such as ITC and Mergenthaler), you will able to obtain screen-compatible font libraries for your screen *and* printer without any Adobe license. As a matter of fact, Microsoft has licensed the Folio technology from Sun for inclusion in Presentation Manager.

The Red Book Has Adobe Seeing Red

Rumors have it that some laser-printer manufacturers are using X/NeWS imaging in their printers driving Canon laser engines instead of Adobe's PostScript! A whisper of a rumor indicated that the printing experts at Linotype are designing a SPARC-based Raster Image Processor (RIP) using X/NeWS instead of PostScript. I wonder if Adobe founders Warnock and Gaescke are banging their heads for publishing the definition of PostScript in their "red book."

I would expect Sun to announce shortly the availability of low-cost, high-resolution laser printers with PostScript . . . er . . . X/NeWS capability. The "rasterization" could be performed by stealing some cycles from a high-powered SPARC on a network and just downloading the bits to a printer. The NeXT laser printer uses this approach by utilizing the local

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



68030 processor to convert the PostScript to raster. My guess is that Sun will price its higher-resolution NeWS laser printer around $1,500 (that is, it'll be cheaper than the NeXT printer).

What's Not Black and White and Read All Over?

Sun's future low-cost monitors, that's what. Considering what Sun V.P. Bill Joy said at a recent talk, Sun is moving away from black-and-white monitors and toward gray-scale monitors. It's no coincidence that the new graphics accelerators (the GX boards) from Sun perform anti-aliasing in hardware. Instead of the physical resolution of the monitor being increased, the "effective" resolution will be raised by having shades of gray. (It's currently more expensive to increase the physical resolution of a monitor than the effective resolution to the human eye. This way, Sun can maintain reasonably low prices.) The NeXT box uses four shades of gray to increase the effective screen resolution.

It's a Sony

Good ol' Japanese engineering from Sony Corporation is rumored to be announcing a 2K-by-2K color Trinitron monitor within six months. Also, look for cut-rate prices on Sony's older 68020 workstations. Sony wants desperately to break into the American workstation market. My contacts in Japan suggest prices in the mid $6,000 range for a complete color system running X Window and BSD 4.3 UNIX.

Sun's Network File System (NFS) is now available for the Microsoft Windows environment. Distinct Corporation (408-741-0781) is offering the Network Windows Development Kit for PCs ($195). Now Windows applications can utilize a UNIX-secure distributed file system via the industry-standard NFS. By using Ethernet and NFS, Windows applications can gain transparent access to any data file on the network that conforms to the NFS specification—essentially, any file on the network. This capability was available before through Sun's PC-NFS product running standard MS-DOS applications, but now Windows has the same ability.

I've designed many DOS-based PC systems using NFS, and it's neat to get files off VMS and MVS machines in a totally application-transparent fashion. Your application doesn't even know whether the file is local or remote. One imaging system that I put together for a client connected an optical disk jukebox with terabytes of data front-ended with a Sun workstation (via NFS, of course) to a 386 box running PC-NFS over standard Ethernet. It was almost ludicrous—but extremely valuable—for the E drive on the PC to have terabytes of storage.

SPARCs Are Flyin'

Cypress Semiconductor announced a new cache-chip set for their implementation of the SPARC architecture. By utilizing their CY7C604 cache controller and the CY7C157 cache RAM, the Cypress SPARC part cranks up to 24 (VAX) MIPS.

Via Via

Fellow SPARC International member Fujitsu Microelectronics has invested quietly in chip-set maker Via Technologies. Apparently, Fujitsu is very interested in producing a SPARC chip set. I talked to a Fujitsu engineer recently about this and was told that Via is targeting a complete, 20-MIPS SPARC workstation at well under $10,000 for the middle of 1990.

Simple Sybase Had a Database

Sybase's new Open Server product will soon be available on a heterogeneous network. That means that a Sybase client program (process) running on a PC, for example, will be able to access a Sybase server running on a Sun Sybase server or a VAX VMS server. Also, the Open Server product is written as an active server, not a passive one, meaning it is both a server and a client. If the targeted server cannot supply the client's request, it will forward the request to another Sybase server on the network. In effect, the server becomes a client.

Rumors have it that Sybase is developing a true object-oriented database. That market is still in its infancy. Object-oriented databases would be perfect for hypertext and hypermedia applications.

Hyper Media

I recently had the opportunity to play around with a NeXT box and watch a slickly produced video from "Mr. Image," Steve Jobs. The tape demonstrated the completely computerized, roboticized manufacturing plant in Fremont, California. According to this media promo, a NeXT box can be built in two minutes.

As far as the Display PostScript–based user interface goes, it was also

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

slick. The gray-scale monitor assisted in contributing to a very sexy interface, and it demonstrated very strong—and admittedly unexpected—performance. The engineers did a very good job with the software. Given that the machine was constantly under the load of "rasterizing" PostScript, it was surprisingly fast. I also saw the inside of the black, one-foot cube and was astonished to see that it was practically empty. Most of the circuitry of the NeXT workstation is on one vertical board. It was clear that Steve and the boys have some plug-in boards up their sleeve. When I asked about color, I got the following response: "The color version is a few years away. But we are now looking at some tremendously innovative and unbelievable monitor technology from Japan."

Hype? I don't know. Certainly Jobs is not going to wait a few years for a color version (that was standard marketing misinformation). I was just very impressed with the user interface. It looked like something that was described at a recent SIGGRAPH conference, though.

I talked to some Sun people at the last X Window Conference at MIT, and they too were impressed with the user interface. They admitted that they were going to incorporate elements of the NeXT interface in some future user interface.

That's Not NeWS to Me

Speaking of the X Window Conference, someone from HP gave a talk on how complicated it was to create simple shadows on X Window. It demonstrated how much work you have to do to get simple drop shadows on "widgets" (X Window with associated semantics). Besides being difficult to create, the run-time effort is really not worth it. It is absurd, since it's so easy to create shadows using the NeWS window system—as a matter of fact, it's trivial.

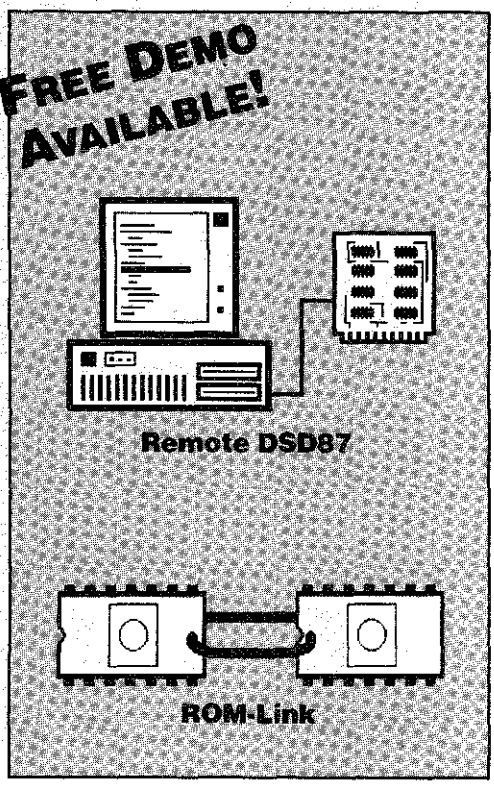
In order to combat MIPS Computer Systems for the lucrative Japanese workstations market, Sun and Fuji Xerox have formed a joint venture, called Unisol Corporation. This new company will develop and market UNIX-based software for the SPARC architecture in Japan.

Quick RISC ASIC

Sony has recently announced that the RISC-based R3000 MIPS chip set will power their future workstations. According to a Sony executive, the MIPS chip was selected for technological and political reasons. One of the most important factors, according to a quote in *EE Times* by the general manager of Sony's workstation division, was "the number of gates in the MIPS design. In the future, the CPU will be just a small portion of an ASIC." Seems like Sony is planning to build application-specific, nonstandard workstations in the future.

From what I've read in the Japanese trade rags, Sony is number one in the Japanese workstation market, with Sun a very close second. Apparently, Sony feels that jumping on the SPARC bandwagon will threaten their position; hence the MIPS agreement. I don't see it that way. Of course, using the SPARC architecture will benefit Sun from a public relations point of view, but look at what Sony could add to a SPARC-based workstation. Sony engineers are video experts, optical disk experts, CD experts, and graphics experts. Imagine an inexpensive SPARC workstation, totally compatible with all the industry-standard software (SPARCware), connected to a low-cost Sony erasable

For the Ultimate in Embedded Systems Programming.



Debug Debug Debug Debug Debug

Remote DSD87 is the best source debugger for embedded development and the only product offering high performance and advanced software In Circuit Emulator capabilities. This package and your compiler form a complete solution for the development of embedded programs. Order now and receive a FREE copy of Native DSD87, a $125 value.

Remote DSD87 (supports both Microsoft and Turbo C) . . . $395
Remote DSD87 OMF (includes Intel OMF support) $795

Link Link Link Link Link Link Link Link

ROM-Link is the first integrated link and locate utility to directly generate COM, EXE, SYS, HEX, or Intel OMF files. The only linker ever guaranteed faster than any present or future competitors. Supports Microsoft debugging information. Receive a FREE librarian and cross reference utility with your order!

ROM-Link $395

Soft Advances
10811 Washington Bl., Ste. 205 • Culver City, CA 90232 • (213) 559-7015

Number **33** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MicroWay Means Numerics!

MicroWay is your best source for the software and hardware you need to get true 32 bit performance from your 386. These include 32-bit tools, such as NDP Fortran, C, and Pascal, and the 32-bit applications that were developed with them (see last paragraph). These products run in protected mode under Unix, Xenix, or Phar Lap extended MS-DOS.

Starting with release 1.4VM, NDP Fortran, C and Pascal not only access 4 gigabytes of memory, but run with Phar Lap's new VMM extension which provides 386 protected mode virtual memory. Now you can run a program with a 30 MB array on a 2 MB system simply by having 30 MB of space on your hard disk.

MicroWay also offers transputer based parallel processing boards and languages that run in an XT, AT, or 386. Each of the T800 RISC processors on these boards packs the power of a 20 MHz 386/1167. Our best selling board, the Quadputer2™, has four T800s and boasts 40 MIPS/6 megaflops of processor throughput.

MicroWay manufactures Weitek 1167 and 3167 coprocessor cards that run with the 80386. Both cards include an 80387 socket. The 1167 is 2 to 4 times faster than the 80387. The 3167 runs 30% faster than the 1167 in double precision. The key to achieving this speed increase is our NDP Fortran or C and the new 32-bit applications that offer Weitek support. Either processor provides a dramatic increase in throughput for graphics intensive applications. These include VersaCad and Hoops 3D graphics, ANVIL 5000 CAD/CAM, SRAC and Swanson Analysis finite element packages, Mathematica and a host of other packages that were recently ported to the 386 using our NDP Fortran and NDP C. **Please call (508) 746-7341 for more information.**

Dr. Robert Atwell, leading defense scientist, calculates that NDP Fortran-386 is saving him $12,000 per month in rentals of VAX hardware and software while doubling his productivity!

Fred Ziegler of AspenTech in Cambridge, Mass. reports, "I ported 900,000 lines of Fortran source in two weeks without a single problem!" AspenTech's Chemical Modeling System is in use on mainframes worldwide and is probably the largest application to ever run on an Intel processor.

Dr. Jerry Ginsberg of Georgia Tech reports, "My problems run a factor of six faster using NDP Fortran-386 on an mW1167 equipped 386/20 than they do on my MicroVAX II."

32-Bit Compilers and Tools

NDP Fortran-386™, NDP C-386™, and **NDP Pascal-386™** compilers generate globally optimized, mainframe quality code. Each runs in 386 protected mode under Unix, Xenix or Phar Lap extended MS-DOS. The memory model employed uses 2 segments, each of which can be up to 4 gigabytes. They generate code for the 80287, 80387, mW3167 or mW1167 and include high speed EGA graphics extensions written in C that perform BASIC-like screen operations. **NDP Fortran-386™** is a full implementation of FORTRAN-77 with Berkeley 4.2 and Fortran-66 extensions. **NDP C-386™** is a full implementation of AT&T's PCC with MS and ANSI extensions. **NDP Pascal-386™** is a full implementation of ANSI/IEEE Pascal, with extensions from C and Berkeley 4.2 Pascal.

NDP Fortran/C/Pascal-386/DOS each $595
NDP Fortran/C/Pascal-386/VM. $695
NDP Fortran/C/Pascal-386/UNIX. $795

Phar Lap 386/VMM extensions are supported by the VM releases of NDP Fortran, C, and Pascal, making it possible to compile and run programs as large as the free space on your hard disk.

Phar Lap Virtual Memory Manager . . $295
Phar Lap Development Tools $495

NDP Windows™ — NDP Windows includes 80 functions that let you create, store, and recall menus and windows. It works with NDP C-386 and drives all the popular graphics adapters . . . . . Library: $125, C Source: $250

NDP Plot™ — Calcomp compatible plot package that is callable from NDP Fortran. It includes drivers for popular plotters and printers. Works with CGA, MDA, EGA and VGA . . . $325

NDP/FFT™ — Includes 40 fast running, hand coded algorithms for single and double dimensioned FFTs which take advantage of the 32-bit addressing of the 386 or your hard disk. Callable from NDP Fortran with mW1167 and 80387 support $250
387FFT for 16-bit compilers $250

NDP to HALO '88 Graphics Interface — Enables you to call graphics routines in HALO '88 from NDP Fortran, C or Pascal. $100

Parallel Processing

Videoputer™ — The highest performance graphics card on the market. Uses a T800 and TI 34010 in conjunction with a 130 MHz Brook-Tree DAC. Includes one MB of system RAM, one meg of video RAM and a library of graphics primitives. Runs standalone or in conjunction with a transputer network and drives 32 and 64 KHz analog monitors $4995

Monoputer™— The world's most popular PC transputer development product now extends the memory available for developing transputer applications from 2 to 16 MB. The board features a DMA bus interface for fast I/O.

Monoputer with T414 (0 MB) $995
Monoputer with T800 (0 MB) $1495

Quadputer™ — This board for the AT or 386 can be purchased with 2, 3 or 4 transputers and 1 or 4 MB of memory per transputer. Two or more Quadputers can be linked together to build networks of up to 100 or more transputers providing mainframe power . . . . . from $3495

Linkputer™ — Uses four Inmos C004 programmable cross bar switches. It allows the user to dynamically change the topology of the processors in a network. Using this board, it is possible to get 100% linkage among eight Quadputers and design larger, custom topologies. . . . . CALL

Transputer Compilers and Applications
These Parallel languages are designed for use with Monoputer2 and Quadputer2.

Logical Systems Parallel C $595
3L Parallel C or 3L Parallel Fortran . . $895
COSMOS/M - Finite element analysis . CALL
ParaSoft: Parallel Environment . . . $500
Performance Monitor . . . $500
C Source Level Debugger $500
T800/NAG™ (See NDP/NAG) $2750

387BASIC™ — Our 16-bit MS compatible compiler introduces numeric register variables to produce the fastest running 80x87 code on the market. $249

NUMERICS HOTLINE
(508) 746-7341

386 Your AT

386/387 Turbo™AT — This board plugs into your 80286 socket, allowing your IBM AT to run 32-bit protected-mode code written for the 80386. Includes an 80387 socket. The most cost-effective AT upgrade!

386/387 Turbo AT/16MHz $495
386/387 Turbo AT/25 MHz $695

Weitek-Based Coprocessor Boards

mW1167™ and **mW3167™** coprocessor boards are built at MicroWay using Weitek components. Each includes an 80387 socket.

mW1167-16 . . . . . $895
mW1167-20 . . . . . $1095
mW1167 Microchannel-16/20 . . . . . CALL
3167-20 . . . . . $995
3167-25 . . . . . $1295
3167-33 . . . . . $1695
mW3167/80387 Board $150

Intel Coprocessors and RAM

8087	$84	**80287-2**	$120
80287-8	$195	**80287-10**	$220
80387-16	$350	**80387-16SX**	$310
80387-20	$400	**80387-25**	$500

80C287A (CMOS) $280
287Turbo-12 (for AT compatibles) $350
RAMpak™ - one meg 32-bit memory module for Compaq 386 20/25 . . . . . $495
RAMpak™ - four meg $1995
256K 80ns DRAM $8.00
256K 100ns DRAM $6.50
256K SIMMs 100ns $99
1 meg SIMMs 100ns $300
(All of our Intel coprocessors include 87Test.)

Multi-User Accelerators

MicroWay's **AT8™** and **AT16™** intelligent serial controllers run 8 to 16 terminals under Unix or Xenix without bogging down your AT, 80386 or PS/2 PC. **AT8**: $895 **AT16**: $1295

12 MHz PC Accelerators

FastCACHE-286 12 MHz $299
SuperCACHE-286 12 MHz $399
FastCACHE-286 9 MHz $199

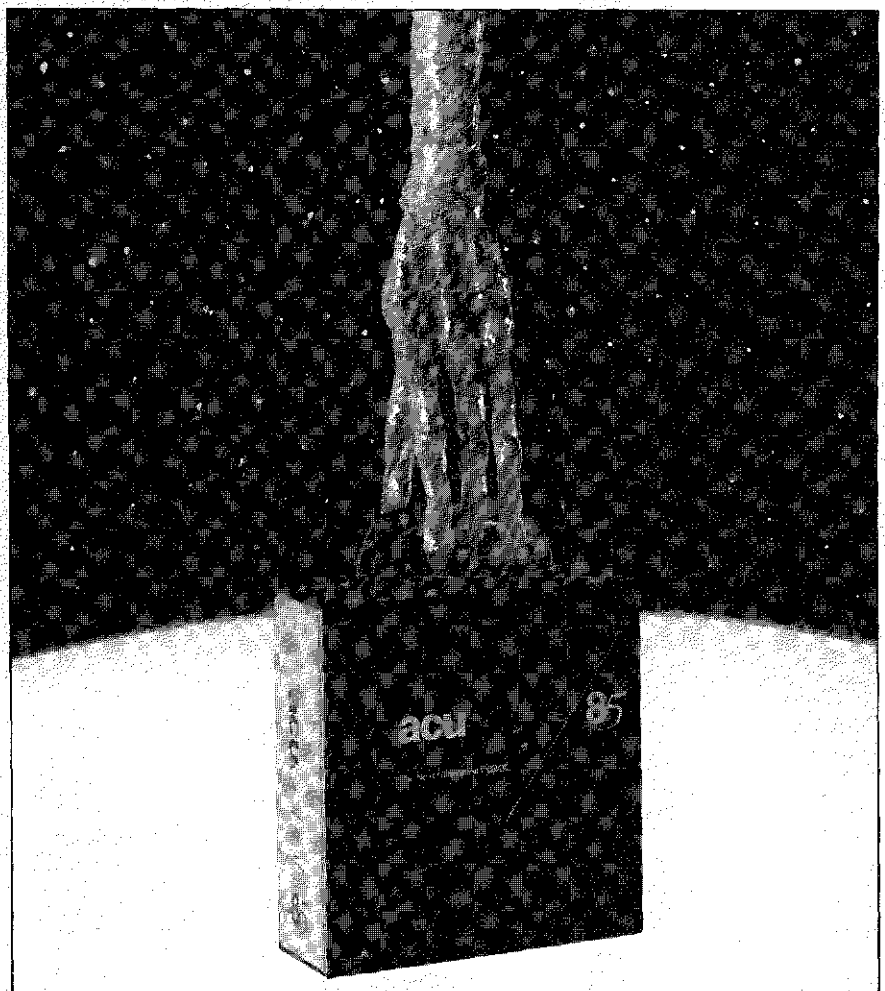
MicroWay

World Leader in PC Numerics

Corporate Headquarters: P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K., 01-541-5466
USA FAX 508-746-4678 Germany 069-75-2023

Number 104 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



INTRODUCING THE MOST PORTABLE COBOL IN THE UNIVERSE

Go where you want to go
Transport COBOL programs to almost any platform—without changing one line of code. Run on AIX, MS-DOS, NETBIOS, UNIX, OS/2, ULTRIX, XENIX and VMS without recompiling. No other COBOL-85 offers such hardware independence, or the instantly-expanded possibilities.

Open new windows in your applications
Create better-looking programs with windows. Add help screens, pop-up/pull-down menus, context-sensitive messages, more. ACUCOBOL-85 has a Display Window verb, including extensions for boxed, reverse-video and pop-up windows in color. Paint special forms on the screen with our Display Line verb. And scroll up/down inside windows.

Don't get lost in space:
ACUCOBOL-85 keeps you on course
ACUCOBOL-85 is NBS certifiable error-free under the High 1974 and Intermediate 1985 ANSI standards. If you lose course, it has its own interactive source code debugger that's fast and easy to use.

Step into the future
Make your job simpler and stay competitive—even ahead of the game. Put the next generation COBOL to work for you today.

Free evaluation copy
Call (619) 271-7097 today for your free evaluation copy of ACUCOBOL-85. And see the shape of things to come.

acucobol inc

Voice: (619) 271-7097 Fax: (619) 566-3071
7950 Silverton Avenue Suite 201 San Diego California 92126

ACUCOBOL-85/the ACUCOBOL logo are trademarks of ACUCOBOL, Incorporated; MS-DOS/XENIX of Microsoft Corporation; NETBIOS/OS/2/AIX of International Business Machines Corporation; VMS/VAX/ULTRIX of Digital Equipment Corporation; UNIX of AT&T Bell Laboratories; RM/COBOL and RM/COBOL-85 of Ryan-McFarland Corporation.

Number **282** on the Reader Service Card

optical disk, using a Sony ultrafast video graphics chip showing real-time video in X Window, utilizing advanced hypermedia applications with Sony CDs, all displayed on a Sony Trinitron. Obviously, Sony puts the package together cheaper than anyone, including Sun. The MIPS chip gives them the bells and whistles but not the meat and potatoes.

By not using the SPARC architecture and leveraging the demand for low-cost, RISC-based, industry-standard software, Sony will not be able to produce low-end workstations easily. I guess they haven't learned their lesson from the Betamax marketing fiasco.

Sun Down?

The SPARC Vendor Council, a non-profit organization of SPARC vendors, has been reincorporated as SPARC International. In addition, to show its independence from the future designs for the SPARC architecture, Sun has surprised everyone and left the group. This implies that the future of the SPARC architecture is now up to SPARC International and not Sun. And to top that off, the SunOS operating system binary and source codes are now available directly from SPARC International as well as from Sun. Phoenix, the BIOS people, were one of the first licensees of SunOS.

DynaMITE DynaSPARC?

Considering the strong ties that Japan's Toshiba Corporation has with Sun, I wouldn't bet against a SPARC laptop in the near future.

Failed Emissions Inspection

Speaking of Toshiba, their 20-MHz, 386-based laptop, the T5200, has finally been shipped. Initially announced in November 1988, the first release of the machine apparently failed the necessary FCC inspection for radio frequency leaks. Now that Toshiba has corrected the problem and the FCC has given the T5200 their blessing, this machine looks like a powerhouse, especially with the 100-megabyte, built-in hard disk and UNIX availability. It should give Compaq a run for their money. Toshiba just has to reduce the price of the machine—$11,000 for a laptop is *too* high, even if it does have a 100-megabyte drive.

Saber Tools Tiger

Saber Software has a fantastic C development environment running on

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

UNIX-based workstations. It provides an interpretive environment that gives the programmer an extremely high degree of process debugging. The Saber environment was used in developing the X Tool Kit intrinsics at DEC and MIT. Being a big advocate of visual programming environments and a former member of a team that developed a similar product, I applaud Saber's efforts (and results).

Apollo Bites the Big One

On the same day as Sun's SPARC-Station announcement, Apollo Computers quietly announced that they will be acquired by Hewlett-Packard. According to the *Wall Street Journal*, this causes HP to jump to first place in the workstation market from third. I'm sorry, but I just do not see the long-term benefits that HP gets from Apollo. Obviously, HP gets the Apollo client base, which gives it some income, but Apollo is not known for their software technology. The only advances that Apollo had recently made were in conjunction with other OSF cohorts, IBM and DEC—and *that* was only because IBM and DEC are extremely concerned about stopping Sun, not because of some significant technology from Apollo.

The only thing that OSF-er HP could be interested in is Apollo's Prism 64-bit proprietary architecture—and that's questionable. Come to think of it, HP's only recent computer-related advances were also in conjunction with other companies: Microsoft for New Wave (I still think that's a stupid name) and PM/X, the Presentation Manager look and feel over the X Window system. This will prove to be a big mistake for HP.

Image Conscious

NEC has a low-cost image-compression engine called the NEC Advanced Compression/Expansion Engine (ACEE). A 1728x2376 black-and-white image can be compressed in .34 second and reconstructed (expanded) in .54 second. It supports some primitive multitasking as well as shrinking and enlarging of the image—seems to be perfect for a PC fax board.

There's a Call on the Line

If you are interested in procedure calls that are extended over a network, contact Netwise, Inc. (303-442-8280). They have a product called RPC TOOL that provides 100 percent of the communication software needed to extend the semantics of an ordinary procedure call over any supported network in a reasonably transparent fashion. It is available for hardware ranging from PCs to mainframes, running UNIX, DOS, OS/2, and VMS on such networks as Novell, TCP/IP, LAN Manager, TLI, and NetBIOS. Future versions will support SNA/APPC.

Smokin' Mirrors

Texas Instruments demonstrated a true 3-D image in real time. Using a technology called multiplanar displays and vibrating flexible mirrors, researchers have been able to produce a virtual volume of 25x25x25 cubic centimeters and 4096x4096 addressability in each plane. This display does not require the viewer to wear glasses like the other technologies but provides very impressive, "true" 3-D laser imaging produced in real time, viewable from all angles and by many viewers. According to the TI researchers, commercial realization is five years away.

Turbo Pascal Tools to Help You Program Faster and Easier

A Multi-User B-Tree Toolkit

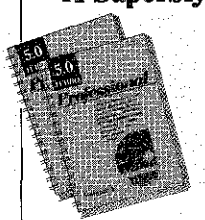
Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

You get: ■ Fixed and variable length records ■ Two billion records per data base ■ Up to 100 indexes per file ■ Flexible record locking ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

B-Tree Filer 5.0 is only $125. (single user)
With network support, only $175.

"A Superbly Crafted Toolbox"



Turbo Professional 5.0 is a library of more than 600 state-of-the-art routines optimized for Turbo Pascal 5.0. It includes complete source code, comprehensive documentation, and powerful and useful demo programs.

You get: ■ TSR management ■ Menus and windows ■ BCD ■ Large arrays and more. *New Routines!* Complete mouse support ■ Window-oriented text editor ■ Scrolling data entry screens ■ Versatile pick lists ■ On-line reference guide, and more.

"Turbo Professional...a superbly crafted toolbox... [makes] the hard stuff so easy."
Kent Porter, Dr. Dobb's Journal, 4/88

Turbo Professional 5.0 is only $125.

Analytical Tools to Help You Write Better Programs

You'll write better Turbo Pascal 5.0 programs easier and faster using the powerful analytical tools of **Turbo Analyst 5.0**. You get an integrated development environment with: ■ File Browser ■ Mouse support ■ Hotkey access to the compiler, debugger and the analytical tools.

The tools include: Pascal formatter, Program structure analyzer, Program lister, three Execution profilers, and Unit disassembler.

"Turbo Analyst...a valuable tool for every Turbo Pascal programmer. The Program Structure Analyzer...a timesaver. The integrated environment...impressive."
Namir Shammas, Turbo Tech Report, 7/88

Turbo Analyst 5.0 is only $99.

Satisfaction guaranteed or your money back within 30 days. Turbo Pascal 4.0 or 5.0 is required. Shipping and taxes prepaid in U.S. and Canada. Elsewhere add $15 per unit. For more information call (408) 438-8608.

TURBO Power

TurboPower Software P.O. Box 66747 Scotts Valley, CA 95066-0747

Call toll-free for credit card orders.
1-800-333-4160
Continental U.S. and Canada

Number **25** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY!

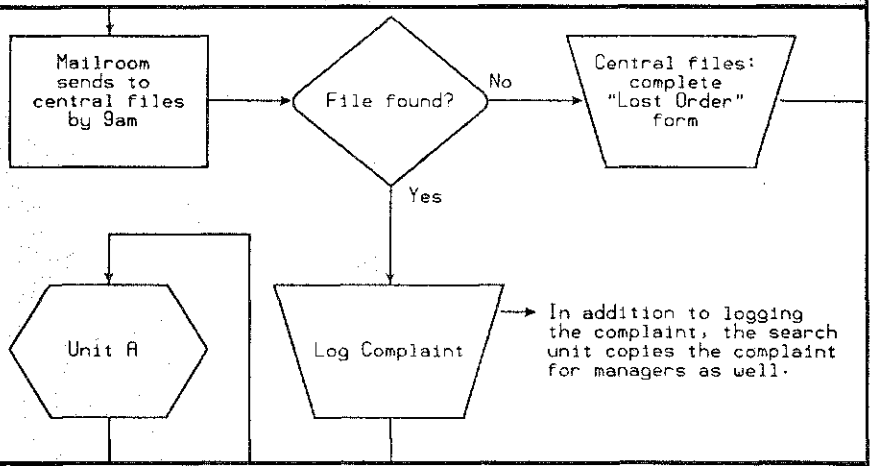
EASYFLOW

An on-screen flowchart processor that knows about flowcharts - not just another "screen draw" program that makes you do most of the work. EASYFLOW is a powerful full-screen graphics program dedicated to flowcharts and organization charts. With it you can quickly compose charts. More important, you can easily modify charts so they are always up to date.

- **Automatic:** Fully automatic text centering within shapes, both horizontally and vertically. Fully automatic line routing & re-routing.
- **Fast:** Written in assembly language for speed.
- **Large:** Charts up to 417 columns wide by 225 lines high. Chart too large for your printer? *EASYFLOW* automatically breaks the chart up & prints it in page size pieces.
- **Standard:** All standard flowcharting shapes included. Custom shapes can be ordered.
- **User friendly:** Don't take our word for it. PC Magazine says *"EASYFLOW lives up to its name. It's hard to imagine any easier and more flexible way to produce basic and even complex flowcharts"*. [March 10th 1987 issue, page 278.]
- **Mouse:** Optional but fully supported.
- **It prints:** On most popular matrix printers including IBM, Epson, Toshiba, HP LaserJet, LaserJet-Plus and many others.
- **It plots:** On HP7440, 7475, 7550, 7570, 7585B and compatible plotters.
- **It works:** We are contractually prevented from mentioning the name of the "big eight" accounting firm that purchased a world-wide site license, but we can tell you that they spent months evaluating all available flowcharting packages before choosing *EASYFLOW*.
- **Rush delivery:** Order by noon today (eastern time) and you'll have it tomorrow. Rush delivery charge is $10.50 (instead of $2.00) and is available only to continental USA and Canada.
- **Documented:** 100 page manual plus over 150 screens of context sensitive help.

EASYFLOW works on IBM PC's, IBM PS/2 and compatibles. Requires 384 K memory, DOS 2.0 or higher and an IBM CGA/EGA/VGA or Hercules monochrome compatible adapter card.

Order direct for only **$149.95** + $2.00 S&H (USA/Canada), $10.00 (foreign). Payment by M.O., cheque, VISA, Mastercard or Company PO.


Mailroom sends to central files by 9am
File found?
No
Central files: complete "Lost Order" form
Yes
Unit A
Log Complaint
In addition to logging the complaint, the search unit copies the complaint for managers as well.

The chart fragment above was produced on an HP LaserJet-Plus and is actual size and unretouched. Publication quality charts like this can be produced using only minutes of preparation time and seconds of print time.

HavenTree Software Limited
PO Box 1093-V
Thousand Island Park, NY 13692

Order Desk: 1-800-267-0668
Information: (613) 544-6035 ext 31
Telefax(G3): (613) 544-9632

Number **154** on the Reader Service Card

Is That a PC in Your Pocket or Are You Just Happy to C Me?

Distributed Information Processing, Ltd., of Guildford, England, has inked a deal with Atari to market a PC that can fit in your pocket. While DIP calls it the Pocket PC, Atari is going to call it the Atari Folio. It is 8 inches by 4 inches and comes with 128K RAM and 256K ROM. The processor is a CMOS 8088, and it is powered by three AA batteries yielding six to eight weeks of battery life. There will be no disk drives, but removable 128K RAM or ROM credit-card-sized modules will be the portable media. Display is 40 characters by 8 lines. DIP is planning a 286 version in the near future.

Quote of the Month

In one of the trade rags, Adrian King, Microsoft's director of product marketing and operating systems, was recently quoted as saying, "Don't expect UNIX to be a competitor in corporate workstations such as you or I might use." He also said, "We'll be far ahead of UNIX with high-capacity graphics applications." Ha.

Making the Rounds

It's kind of funny that I started this column on a Wyse 286 using Wordstar, "unWordstared" it and uploaded it to a Sun workstation and the VI editor, then transferred it to a Toshiba 1100 laptop (for my hospital gig) using WordPerfect, went back to the Sun, downloaded it to a generic 386 box, and finally finished it on an XT clone using Microsoft Word. PJ

Frank D. Greco, president of Crossroads Technologies, Inc., performs industry analysis, systems consulting and training, and technological forecasting for Fortune 500 companies. Readers can contact Frank at Crossroads Technologies, Inc., P.O. Box 530, Fanwood, NJ 07023, 201-754-7820.

Mktg/Sales Team For Hire

Watch Your Sales Soar!
We help software publishers and hardware manufacturers get the most out of their limited marketing dollars. Let *Techsel* help you market/sell your product - we can *guarantee* your satisfaction. Call Ed Harms or Stewart Gittelman at 516-277-7000.
***Techsel* Corporation**

Number **264** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

epsilon
programmer's editor

State of the Art editor for DOS, OS/2, Xenix & Unix for PC/XT/AT's

Version 4.0 New!

1988 BYTE AWARD OF DISTINCTION

". . . these products–every one of them–are ones we wouldn't hesitate for a moment to recommend." (Byte, Jan. 1989)

Unrivaled Undo AND Redo

Epsilon gives you the most complete Undo and Redo of any PC programmer's editor. You can undo any sequence of changes with a touch of the Undo key. In fact, you can undo the last few hundred or so operations–you set the limit. And unlike the "full" undo offered by some other editors, Epsilon has Redo, so you can undo your undo's. You can even redo your undo's, and undo your redo's! This flexibility keeps you from losing your work by undoing too much.

Ultimate Customizability . . .

Epsilon features the most powerful extension language of any PC editor. We call it EEL, for Epsilon Extension Language. EEL gives you the rich syntax and data types of C, so you can easily extend or customize the editor to suite your taste. EEL is a real language, not an afterthought. In fact, all of Epsilon's commands are written in EEL, and you get complete source code to all the commands!

". . .but the pièce de résistance is EEL, a C-like extension language that comes with its own compiler. Lugaru Software has implemented Epsilon's entire user-interface in EEL so you can, for example, create your own syntax-sensitive modes or program the help system to browse through your source libraries." (Byte, Jan. 1989)

- C-like extension language, called EEL
- Source code to all commands
- Convenient multilevel undo plus total redo
- Scans compiler errors (customizable)
- Swaps out of memory for big compilers
- Unlimited file size, number of files
- On-line tutorial for quick startup
- As many windows as will fit on the screen
- Supports large displays (like VGA & EGA)
- Context sensitive help autoconfigures
- Tags package indexes subroutine definitions
- Files can be larger than available memory
- Fully utilizes EMS memory
- Interactively customizable keyboard
- EMACS-style command set
- Record keystroke sequences
- Multitasks compilers, even under DOS
- Language support (C, Lisp, etc.)
- Completion on file names, commands, tags, ...
- Full command line editing
- Optional backups
- Regular expression search and tagged replace
- Source level debugger & profiler for EEL
- Do interrupts from EEL (DOS)
- Call any dynamic-link library for EEL (OS/2)
- Saves deleted text (n times)
- Navigate your directory structure
- Not copy protected, for your convenience

. . . & True Multitasking Compiles!

Epsilon's exclusive multitasking facility lets you run your compilers, utilities, or even the DOS command processor, in an Epsilon window, WHILE YOU EDIT. You'll be amazed to see your compiler chugging away while you continue to edit your files. No more dead compile time! If any compiler errors appear, just touch a key and Epsilon parses the error message, then jumps right to the offending line–while your compiler is still running! Don't be fooled by misleading claims from other vendors: Epsilon is the ONLY editor for DOS with this feature! Of course, Epsilon's multitasking also works under OS/2, Xenix, and Unix.

"This feature is especially useful in a programming environment because compilers also can be run in the process buffer. . . If the screen is split into multiple windows, one of which is the process window, the user can see the compilation output while continuing to edit... A single command takes the user to the next error line in a source file and displays the error on the bottom of the screen." (PC Tech Journal, May 1987)

Ask about our free trial!

$195 direct

60-day money-back guarantee!

Order Hotline: (412) 421-5911

FAX: (412) 421-6371

Lugaru Software Ltd. / 5843 Forbes Avenue / Pittsburgh, PA 15217

Number **271** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Measuring Performance

by

Michael Abrash

Installment 13: In which we learn that the performance of assembly language code is not so easily predicted and come away with a potent tool for evaluating optimizations and generating high-performance code. (This article is adapted from The Zen of Assembler, *by permission of Scott, Foresman and Company.)*

In this installment, we're going to take a side trip away from pure graphics—but not *too* far away. As you may have noticed, I'm more than a little interested in performance. When it comes to graphics programming for the PC, there's really no choice but to make performance a top priority; the hardware just isn't fast enough to compensate for mediocre code. Someday, the 8514/A, the 34010, and their successors may change that . . . but for now the VGA is the mainstream, and that means that good graphics code requires solid assembly language skills and a thorough understanding of the hardware.

Now, don't get me wrong—I *like* programming the VGA. It's an ever-fascinating puzzle, and it presents fresh challenges even after years of programming. That doesn't change the fact that the VGA absolutely requires top-notch assembly language code.

Which brings us to today's topic: How can you *know* what's good code? The true nature of code performance on the PC is more complex than you would ever imagine. Given all the factors affecting performance—including instruction cycle times, instruction prefetching, memory refresh, and video wait states, not to mention the newfangled memory architectures of the 80386 machines—how can you tell whether your code runs faster or slower after you tweak it? The question matters a *lot* to graphics programmers, but the answer isn't easy to find.

Actually, there are two answers to the "true performance" question—a long answer and a short answer. The long answer can be found in my two-volume book, *The Zen of Assembler*, the first volume of which should be out shortly after you read this. As you might guess, the long answer isn't something you're going to learn in a day or from one article. The short answer, on the other hand, can be learned in a minute. That answer: Measure it.

The truth is that no one—not even the people at Intel who design the chips—can tell how fast PC code runs without measuring it. You can make intelligent guesses, establish rules of thumb, try out dramatically different approaches, search for better algorithms, and fit your code precisely to each application, with no wastage—but when push comes to shove, you won't know whether you've made your code better or worse until you see how long it takes to run.

While that may not seem logical (and may, in its nondeterminism, distress those of you who like to think of computers as orderly, predictable machines), it is true. In case you doubt that, try to guess how long the following code, a typical code for adding the Y coordinate into the display-memory offset of a pixel, will take to run on a standard IBM PC:

```
shr   dx,1
shr   dx,1
shr   dx,1
add   ax,dx
```

What was your guess? Probably nine cycles, which is the total of the execution times for the above instructions that you'll find in Intel's manuals. That's a good guess . . . but wrong, *wrong*, WRONG!

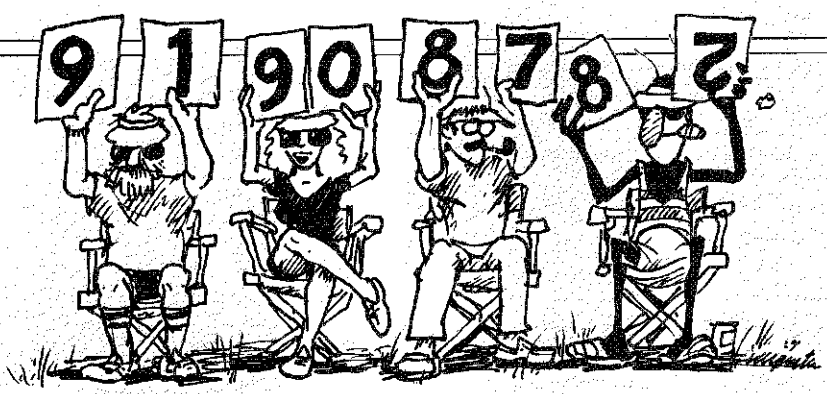
I can't tell you exactly how fast the above code will run; that depends primarily on how full the prefetch queue is when the code starts. I *can* tell you, however, that the above code won't ever run in less than 16 cycles and might well take 32-plus cycles if the prefetch queue starts out empty. (Later, we'll verify that this phenomenon does indeed exist.) From 9 cycles to 16 cycles to 32 cycles and more . . . I hope you see now why you must measure your code's performance in order to know how good it is. If not, check out "More Optimizing for Speed," in *PJ* 4.4, and my new book; you'll come away a believer, I assure you.

Software Timers

Software timers for the PC have appeared in print before. In particular, "Programming Insight: High-Performance Software Analysis on the IBM PC," by Byron Sheppard (*BYTE*, January 1987), was a seminal article on the subject. My timer operates on the same principle as Sheppard's but with significant differences. My timer disables interrupts, thereby providing more-accurate results, and it is limited to a measurement period of about 54 milliseconds (ms). I do have a version that can measure longer periods, but there isn't room to include it here. On balance, my timer is best suited to detailed analysis of relatively small sections of code, while longer-period timers are better suited to analysis of overall program performance. Since we're most concerned about wringing every cycle of performance out of graphics code in "On Graphics," the timer presented here should serve us well.

The most accurate way to measure performance is with expensive hardware, but perfectly good measurements at no cost can be made with the PC's 8253 timer chip, which counts at a rate of slightly over one million times per second. The 8253 can be started at the beginning of a block of code and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



stopped at the end of the block, with the resulting count indicating how long the code took to execute to a resolution of no worse than 1.7 microseconds (µs).

Starting the Timer

Listing 1 shows 8253-based timing software consisting of three subroutines: **TimerOn**, **TimerOff**, and **TimerReport**. **TimerOn** is called at the start of a segment of code to be timed. It saves the context of the calling code, sets the 8253 timer to mode 2 (divide-by-N mode), disables interrupts, sets the initial 8253 count to zero, restores the context of the calling code, and returns.

Two aspects of **TimerOn** are worth discussing further. One point of interest is that **TimerOn** disables interrupts. (**TimerOff** later restores interrupts to the state they were in when **TimerOn** was called.) Were interrupts not disabled by **TimerOn**, keyboard, mouse, timer, and other interrupts could occur during the timing interval, and the time required to service those interrupts would incorrectly and erratically appear to be part of the execution time of the code being measured. As a result, code timed with the timer should not expect any hardware interrupts to occur between a call to **TimerOn** and the subsequent call to **TimerOff** and should not enable interrupts during that time.

Time and the PC

A second interesting point about **TimerOn** is that it may introduce some small inaccuracy into the system clock time whenever it is called. To understand why this is so, we need to examine the way in which both the 8253 and the PC's system clock (which keeps the current time) work.

The 8253 actually has three timers built in, as shown in the figure. All three timers are driven by the system board's 14.31818-MHz crystal, divided by 12 to yield a 1.19318-MHz clock to the timer; so the timers count once every 838 nanoseconds. Each of the three timers counts down in a programmable way, generating a signal on its output pin when it counts down to zero. Each timer is capable of being halted at any time via a gate pin. These timers are inherently very flexible timing devices; unfortunately, much of that flexibility depends on how the timers are connected to external circuitry, and in the PC the timers are connected with specific purposes in mind. Timer 2 drives the speaker, although it can be used for other timing purposes when the speaker is not in use. As shown in the figure, timer 2 is the only timer with a programmable gate input in the PC; that is, timer 2 is the only timer that can be started and stopped under program control. On the other hand, the *output* of timer 2 is connected to nothing other than the speaker. In particular, timer 2 cannot generate an interrupt to get the 8088's attention.

Timer 1 is dedicated to providing dynamic RAM refresh and should not be tampered with under any circumstances lest system crashes result.

Finally, timer 0 is used to drive the system clock. As programmed by the BIOS at power-up, every 65,536 (64K) counts, or 54.925 ms, timer 0 generates a rising edge on its output line. This line is connected to the hardware interrupt 0 (IRQ0) line on the system board; so every 54.925 milliseconds, timer 0 causes hardware interrupt 0 to occur. The interrupt vector for IRQ0 is set by the BIOS at power-up to point to a BIOS routine that maintains a time-of-day count.

Each timer channel of the 8253 can operate in any of six modes. Timer 0 normally operates in mode 3, square-wave mode. In square-wave mode, the

Random Notes

For those of you who need to measure code performance but want a more polished package than the timer, I've heard good things about Paradigm Systems' products. I'm in the process of trying Paradigm's software for myself; by the next installment, I should be able to report back on my experience. (Paradigm Systems Inc., P.O. Box 152, Milford, MA 01757, 508-478-0499)

Another report from the software front: I've been pleasantly surprised by Magna Carta Software's C Windows Toolkit, a text-mode windowing, menuing, all-around, useful screen-management library. I'm from the do-it-yourself-if-you-want-it-done-right school; generally, I've found that canned software (especially video software) runs slowly and unreliably, and that's all the more frustrating when you can't look at—much less change—the code.

Well, C Windows Toolkit is plenty fast enough (being a text-mode-only program helps in the speed department), and it comes with full source code. I haven't had to look at the source code, since I've been perfectly happy with the library as is, but it's nice to know it's there. And while I surely could have written the screen-management code I needed myself, I didn't have to, thereby saving a *lot* of time with no loss in program quality. At $99.95, C Windows Toolkit is more than reasonably priced. Recommended. (Magna Carta Software, P.O. Box 475594, Garland, TX 75047, 214-226-6909)

Finally, I'm asked from time to time what software I use to create figures and diagrams—an interesting question, given that as someone who spends his time creating sophisticated graphics programs, I'm bound to be picky about graphics tools. The truth is that I've tried many paint-and-draw packages over the past few years, and none of them really lived up to my expectations. Finally, however, I've found a program that's a winner, and it'll take one heck of a program to get me to switch again. Micrografx Designer is *superb* (although at $695, it's rather pricey). I'm sure I haven't come close to tapping the full power of Designer—I'm a programmer, not an artist or draftsman—but it does everything I need and does it beautifully. Put this package together with a 386 and a Super VGA running at 800x600 resolution, and you've got an astonishingly good drawing system. (Micrografx, 1303 Arapaho, Richardson, TX 75081, 214-234-1769)

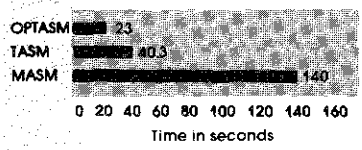
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE PROGRAMMER'S CHOICE

OPTASM
- fastest assembler available
- eliminates phase errors
- never inserts spurious NOPs
- complete TSR online help
- includes a linker and debugger
- "OPTASM is clearly technically superior to MASM" *Computer Language - Dec 88*

$125

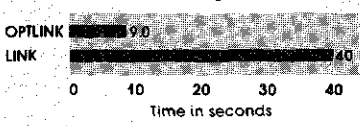
ASSEMBLING **BGIDEMO** ON PC



OPTLINK
- fastest linker available
- generates EXE, COM or SYS
- generates global XREF
- supports CodeView info
- works with MS C5.1

$125

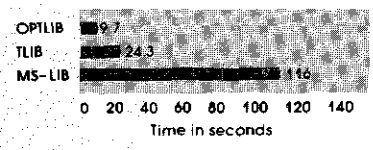
LINKING **BRIEF**



OPTLIB
- 10 times faster than LIB
- * and ? in filenames
- time-based replace
- LIB-compatible syntax
- rename PUBLICs and EXTRNs

$49

BUILD 670K .LIB FILE



OPTDEBUG
- source level - supports CV info
- online help system
- reverse execution
- 50k COM file
- debug EXEs, COMs and TSRs

$125

SLR SYSTEMS
1622 N. Main Street Butler, PA 16001
412-282-0864 TELEX 559215
800-833-3061 (USA)
Call today to order!

Number **165** on the Reader Service Card

initial count is counted down two at a time; when the count reaches zero, the output state is changed. The initial count is again counted down two at a time, and the output state is toggled back when the count reaches zero. The result is a square wave that changes state more slowly than the input clock by a factor of the initial count. In its normal mode of operation, timer 0 generates an output pulse that is low for about 27.5 ms and high for about 27.5 ms; this pulse is sent to the 8259 interrupt controller, and its rising edge generates a timer interrupt once every 54.925 ms.

Square-wave mode is not very useful for precision timing: For each timer interrupt, it counts down by two, rendering exact timings impossible. Fortunately, the 8253 offers another timer mode, mode 2 (divide-by-N mode), which is both a good substitute for square-wave mode and a perfect mode for precision timing.

Divide-by-N mode counts down by 1 from the initial count. When the count reaches zero, the timer turns over and starts counting down again, and a low pulse is generated for a single clock period. While the low level is not held for nearly so long as in square-wave mode, it doesn't matter, since the 8259 interrupt controller is configured in the PC to be edge triggered and hence cares only about the existence of a pulse from timer 0, not the duration of the pulse. As a result, timer 0 continues to generate timer interrupts in divide-by-N mode, and the system clock continues to maintain good time.

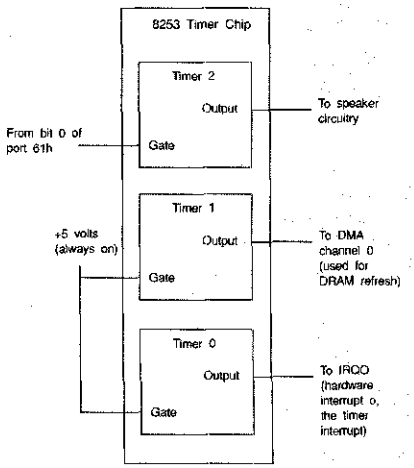
Why not use timer 2 instead of timer 0 for precision timing? After all, timer 2 has a programmable gate input and isn't used for anything but sound generation. The problem with timer 2 is that its output can't generate an interrupt; in fact, timer 2 can't do anything but drive the speaker. We need the interrupt generated by the output of timer 0 to tell us when the count has overflowed. The timer interrupt also makes it possible to time much longer periods than the timer shown in Listing 1 supports.

In fact, the timer shown in Listing 1 can time intervals of only up to about 54 ms in length, since that is the period that can be measured by timer 0 before its count turns over and repeats. Fifty-four milliseconds may not seem like a very long time, but an 8088 can perform more than 1000 divides in that time, and division is the instruction the 8088 performs most slowly. If a measured period turns out to be longer than 54 ms (that is, if timer 0 has counted down and turned over), the timer will display a message to that effect.

The timer determines whether timer 0 has turned over by checking to see whether an IRQ0 interrupt is pending. (Remember, interrupts are off while the timer runs, so the timer interrupt cannot be recognized until the timer stops and enables interrupts.) If an IRQ0 interrupt is pending, then timer 0 has turned over and generated a timer interrupt. Recall that **TimerOn** sets timer 0 to zero, in order to allow for the longest possible period—about 54 ms—before timer 0 reaches zero and generates the timer interrupt.

Now we're ready to look at the ways in which the timer can introduce inaccuracy into the system clock. Since timer 0 is initially set to 0 by the timer, and since the system clock ticks only when timer 0 counts off 54.925 ms and reaches zero again, an average inaccuracy of one-half of 54.925 ms, or about 27.5 ms, is incurred each time the timer is started. In addition, a timer interrupt is generated when timer 0 is switched from mode 3 to mode 2, advancing the system clock by up to 54.925 ms, although this happens only the first time the timer is run after a warm or cold boot. Finally, up to 54.925 ms can again be lost when **TimerOff** is called, since that routine again sets the timer count to zero. Net result: the system clock will run up to 110 ms (about a ninth of a second) slow each time the timer is used.

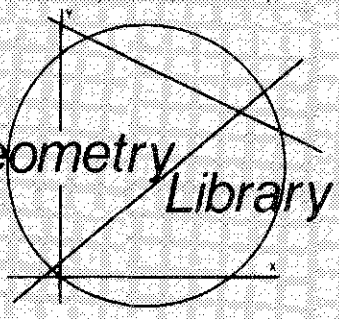
Potentially far greater inaccuracy can be incurred by timing code that takes longer than about 108 ms to



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

*"The cost involved, in writing just one **TurboGeometry** routine, is more than the price of the library itself".*

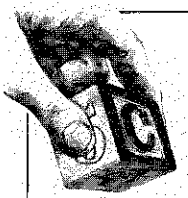
Are you programming or planning to program in CAD/CAM or graphics applications? Many hours, even days can be spent in writing and debugging geometric routines. ***Turbo-Geometry library*** can relieve you of these time consuming tasks. There are over 150 routines in the library. Some of the areas covered are: Equations of lines, circles, areas and planes • Intersection of lines, circles, areas and planes • 2 and 3 dimensional transforms • Decomposition of polygon • Hidden line removal • Volume and area calculations • Perspectives • Polygon line clipping • And many more! Supported by example programs, and a 400 page manual. Source code included. 30 day guarantee. Need IBM PC/COMP, MS/PC DOS 2.0+. Turbo Geometry is available in Turbo C, Turbo Pascal 4.0, MS C or Turbo Pascal MAC. $149.95 plus $5.00 S&H. Foreign orders add $11.00 for S&H. VISA, MC, Chk., MO. To order, call or write.



Disk Software, Inc. • 2116 E. Arapaho, #487
Richardson, TX 75081 • Tel #214-423-7288

*"Give yourself a break. Get **TurboGeometry** Today"*

Number **229** on the Reader Service Card



Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

Call or write today for a free brochure and the latest list of Inside! products.

To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757

PARADIGM SYSTEMS INCORPORATED

Inside! is a trademark of Paradigm Systems.

Contact this advertiser directly

EVERLOCK

COPY PROTECTION

Designed for clone compatibility & strength

- Thwarts ALL Bit-copy Software
- Protect any COM/EXE w/o Source changes
- Shut down Debug Tracing & Disassemblers
- Supports all Hard & floppy Disk formats
- Restrict or allow Network Installs
- Limit program use by Date or #Executes
- Remotely reset Installs, Date or #Executes
- No damaged media or I/O plugs

EVERTRAK

SECURITY

If you HATE Copy Protection, but still want to protect your program from hackers and reverse engineering, **EVERTRAK** allows you to:

- Protect any COM/EXE w/o Source changes
- Shut down DEBUG systems and disassemblers
- Embed/display hacker-proof Customer information
- Restrict use by date-range and/or media-type
- Track unauthorized copies back to their source
- Totally invisible to End Users

For IBM & Compatibles. 30 day money back guarantee.
Free information and demonstration disk available.

TeleCheck VISA MasterCard

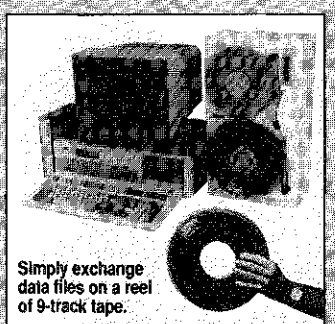
Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644 • (816) 776-2700 • FAX (816) 776-8398

Number **51** on the Reader Service Card

Desktop 9-Track Tape Subsystem

Now, 9-track tape lets your micro exchange data with minis and mainframes.



Simply exchange data files on a reel of 9-track tape.

DEC HP ALTOS NCR — IBM MAINFRAMES — AT&T UNISYS APOLLO SUN

9-TRACK is the first choice for file interchange among data processing professionals. Now, Qualstar's low cost 1/2-inch 9-track *Ministreamer* tape systems bring full ANSI data interchange to IBM PCs or Macintosh, giving your micro the freedom to exchange data files with nearly any mainframe or minicomputer in the world.

Available in both 7" and 10-1/2" versions, compact Qualstar tape drives can sit on your desktop, *using less space than an ordinary sheet of paper.* Systems include DOS or XENIX compatible software, coupler card and cables. High reliability 1600 or 6250 BPI capability may be used for disk backup as well as data interchange. Discover the big advantage 9-track tape has over other micro/mainframe links.

Call us today!

FOR DETAILS AND TO ORDER:
FAX (818) 882-4081
PHONE (818) 882-5822

#1 Selling 9-Track Systems on the Desktop

QUALSTAR

9621 Irondale Ave., Chatsworth, CA 91311

©1988 Qualstar Corp. All product and company names and trademarks are the exclusive property of their respective owners.

Number **97** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

execute. Recall that all interrupts, including the timer interrupt, are disabled while timing code with the timer. The 8259 interrupt controller is capable of remembering at most one pending timer interrupt; so all timer interrupts after the first one during any given timing interval are ignored. Consequently, the system clock falls behind by 54.925 ms each additional time timer 0 turns over while measuring code performance.

The timer's effects on the system time aren't a matter for great concern, as they are temporary, lasting only until the next warm or cold boot. Besides, even heavy use of the timer usually makes the system clock slow by a total of a few seconds at most, unless code that takes much longer than about 54 ms to run is timed (in which case the timer will notify you that the code is too long to time). Nonetheless, it's a good idea to reboot your computer at the end of each session with the timer in order to make sure that the system clock is correct.

Stopping the Timer

TimerOff is called at the end of a segment of code to be timed, at some point after **TimerOn** is called. **TimerOff** saves the context of the calling program, latches and reads the timer 0 count, converts that count from the countdown value that the timer maintains to the number of counts elapsed since **TimerOn** was called, and stores the result. Immediately after latching the timer 0 count—and before enabling interrupts—**TimerOff** checks the 8259 interrupt controller to see if there is a timer interrupt pending and, if there is one, sets a flag to mark that the timer overflowed.

After that, **TimerOff** executes just the overhead code of **TimerOn** and **TimerOff** 16 times, averaging and saving the results to determine how many of the counts were incurred by the overhead of the timer rather than by the code being timed.

Finally, **TimerOff** restores the context of the calling program (including the state of the interrupt flag that was in effect when **TimerOn** was called to start timing) and returns.

One interesting aspect of **TimerOff** is the manner in which timer 0 is stopped to read the timer count. We don't actually have to stop timer 0 to read the count; the 8253 provides a special latched read feature for the specific purpose of reading the count while a time is running. (That's a good thing, too; we couldn't very well stop timer 0 if we wanted to, since its gate input isn't connected.) We simply tell the 8253 to latch the current count, and the 8253 does so without breaking stride.

Reporting Timing Results

TimerReport is called after both **TimerOn** and **TimerOff** have been called, to display the results obtained by those two routines. **TimerReport** first checks to see whether the timer overflowed before **TimerOff** was called. If overflow did occur, **TimerOff** prints a message to that effect and returns; otherwise, **TimerReport** subtracts the reference count (representing the overhead of the timer) from the count measured between the calls to **TimerOn** and **TimerOff**, converts the result from timer counts to microseconds, and prints the resulting time in microseconds to the standard output.

Note that **TimerReport** need not be called immediately after **TimerOff**. In fact, after a given call to **TimerOff**, **TimerReport** can be called at any time right up until the next call to **TimerOff**.

You may want to use the timer to measure several portions of a program while it executes normally, in which case it may not be desirable to have the text printed by **TimerReport** interfere with the program's normal display. You could handle this by removing the invocations of the DOS print string function, instead running the program under a debugger that supports screen flipping (such as Turbo Debugger, Symdeb, or Codeview), placing a breakpoint at the start of **TimerReport**, and directly observing the count in microseconds as **TimerReport** calculates it. Alternatively, you could print the output on a printer, or send it over a serial port.

Notes on the Timer

As implemented in Listing 1, the timer subroutines are near-callable from any assembly language code running in segment **Code**. The timer subroutines can, however, be called from any assembly or high-level language code that generates OBJ files that are compatible with the Microsoft Linker simply by modifying the segment that the timer code runs in to match the segment used by the code being timed. All three subroutines preserve all registers and all flags except the interrupt flag, so calls to these routines are transparent to the calling code.

If you change the timer routines to far procedures in order to call them from a program using the large code model, be sure to make *all* the routines far, including **ReferenceTimerOn** and **ReferenceTimerOff**. Otherwise, the reference routines won't reflect the true overhead incurred by starting and stopping the timer.

Please be aware that the inaccuracy that the timer can introduce into the system-clock time does not in any way affect the accuracy of the performance measurements reported by the timer itself. The 8253 counts once every 838 nanoseconds (ns), giving us a count resolution no worse than about 1.7 µs (838ns * 2). However, factors such as the prefetch queue and video wait states, as well as what seems to be some variation in the time it takes the 8253 to respond to the load of a new count, make it more accurate to describe the timer as measuring code performance with an accuracy of better than 10 µs.

The timer works on all PC-compatible computers I've tried it on, including XTs, ATs, PS/2s, and 80386-based AT-compatible machines.

A Sample Use of the Timer

Listing 2 shows a sample use of the timer. This listing measures the time required to execute the four-instruction sequence I used as an example at the start of the article. Note that Listing 2 calls **TimerOn** to start timing, performs 1000 repetitions of the sample sequence in a row, calls **TimerOff** to end timing, and finally calls **TimerReport** to display the execution time of the test code.

When listings 1 and 2 are assembled, linked, and run on a PC, the timer reports an execution time of 7.242 ms, or about 7.24 µs for each four-instruction sequence. (No matter how many repetitions of a given instruction are timed, there's just too much noise in the timing process—given dynamic RAM refresh, the variability of the internal state of the 8088 at the start of timing, and the like—for that last digit to have any significance.) Given the PC's 4.77-MHz clock, this works out to about 34.5 cycles for the four instructions, which, as I promised at the outset, is a good deal longer than Intel's specified nine-cycle execution time. Fear not, the timer is right—that sequence really does take 34.5 cycles as used in Listing 2.

You can measure the performance of any code sequence simply by substituting that code for the code between the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

calls to **TimerOn** and **TimerOff** in Listing 2. If you wish to time code directly in place in your programs, just insert calls to **TimerOn**, **TimerOff**, and **TimerReport** in the appropriate places and link Listing 1 to your program.

In Conclusion

And that's the timer. It's a *very* handy tool for fine-tuning your loops and time-critical code. (Don't waste your time tweaking noncritical code, though; there's too much code in this world that desperately needs optimizing to waste time on code that doesn't.) Useful as it is, though, tweaking case-by-case performance isn't the ultimate use of the timer; the real excitement lies in the rules about crafting high-performance code that we can develop with the timer. That's a big topic, though—the long answer, as I said at the outset. At least now we have the short answer nailed to the wall: How do you know how good your code is? Measure it with the timer.

I'd be interested to hear whether you'd prefer that I spend more time on graphics or on high-performance assembly language programming from now on. Not that the two are mutually exclusive, by any means—we've been dealing with high-performance assembly language since the first installment of "On Graphics"—but there are many assembly language topics that have nothing to do with graphics, such as searching, sorting, array and string manipulation, and the like. At the other end of the spectrum, I've been considering an in-depth discussion of animation on the VGA.

I'm happy to go either way—so let me know what *you* need.

At any rate, next issue we'll return to pure graphics, probably with a discussion of 360x480, 256-color graphics mode and maybe with a discussion of VGA split-screen panning, if there's room.

See you then. PJ

Michael Abrash is a Senior Software Engineer for Orion Instruments, Inc., of Redwood City, California, where he is working on the design of advanced PC-based test instrumentation. Readers may write to Michael at P.O. Box 39051, Mountain View, CA 94039, or drop him a line via MCI Mail, user name MABRASH.

Code follows.

POWER GRAPHICS FOR YOUR PC!

MetaWINDOW

NO ROYALTIES!

PC TECH JOURNAL
Product of the Month
"Awarded for excellence in product design and development"

Developing a graphics application? Move to the peak of performance and graphics functionality with MetaWINDOW! Built on state of the art "object oriented" display flexibility, now you can take advantage of powerful advanced graphics capabilities like those used on the Macintosh, Sun and Apollo workstations. Using MetaWINDOW you can break the bottleneck between limited function low-level graphic primitive libraries and slow, bulky hard-coded window managers.

MetaWINDOW provides an advanced set of graphic drawing functions, dynamic runtime support for a full range of graphic peripherals, plus unmatched performance for designing multi-window desktop applications.

Awarded Top Honors!

PC Tech Journal, Jul '85
Product of the Month - *"... a technological tour de force for fast PC graphics." "... one high-powered piece of artillery."*

Computer Language, Oct '87 - *"... surprisingly powerful for the cost." Independently benchmarked the fastest of all graphics systems reviewed - faster than Halo, over 3 times faster than GSS*CGI.*

Turbo Technix, Feb '88
"... Customer support is excellent. I've called Metagraphics several times and have gotten prompt, accurate help each time. Metagraphics has a winner with TurboWINDOW."

Superior Functionality!

- Macintosh QuickDraw function superset - *simplifies porting PC & Mac applications.*
- PostScript & X-Window compatible drawing attributes - the defacto graphics *standard for PC X-Window servers.*
- Supports both bitmap and outline fonts - *dynamically face fonts for bold, italic, underline or strike-out effects.*
- Full 16 function "rasterOp" flexibility - *easily write, erase, rubberband or drag: lines, text, icons, bit images and complex objects.*
- Dynamic recognition and run-time support for 80+ popular graphics cards - *eliminates multitudes of clumsy device drivers and bothersome configuration sessions - true device independence at last!*
- Virtual images - *create and manipulate graphic images in local or expanded memory up to 8 megabytes in size.*
- *... and the list goes on!*

Rated #1 by Users!

J. Kayser, CMSI -
"MetaWINDOW is the best graphics package I've seen. Since I bought my PC, I've spent over $2500 on software. I've bought bad software, mediocre software, and good software. There's not much software I call excellent, but MetaWINDOW is excellent!"

David Gray, LUCAS LABS -
"I would also like to take this time to compliment you on providing such a fantastic graphics package. We have used several packages previously including Turbo graphics and Halo graphics and found none that compare to the versatile, high performance graphics package that your company offers. We have also found your technical support staff to be one of the best we have ever dealt with."

John Weaver, Chairperson
Department of Mathematical Sciences
WEST CHESTER UNIVERSITY -
"We have been using MetaWINDOW for our graphics instruction for the past several semesters and are pleased with the package. This past summer we upgraded one system to MetaWINDOW version 3 and were so impressed that we want to upgrade all systems prior to the Spring semester. This is a first class package and I think it is the best for teaching computer graphics."

Reach For The Future!

MetaWINDOW - $195
Advanced Graphics Development Toolkit
MetaWINDOW comes complete with language bindings for 20 popular C, Pascal and Fortran compilers, plus dynamic runtime support for over 80 graphics adaptors and input devices.
4 disks, 3 260 page manuals

TurboWINDOW/C - $95
TurboWINDOW/Pascal - $95
QuickWINDOW/C - $95
All the features of MetaWINDOW for Borland Turbo C, Turbo Pascal or Microsoft C.

NEW - FontWINDOW - $95
MetaWINDOW font/icon editor plus 100 fonts.

NEW - MetaWINDOW/Premium - $495
Power graphics for new high-res, high performance TI34010 based graphics cards!
(Licensing required for redistribution of MetaWINDOW/Premium)

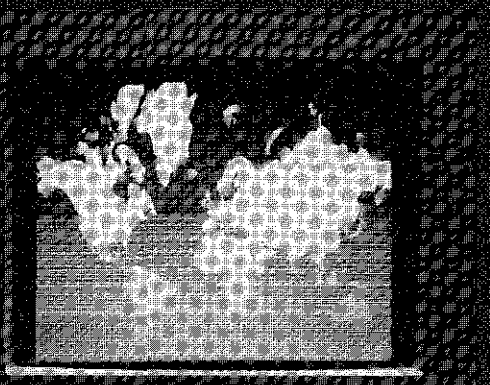
30 Day Money Back Guarantee
ORDER TODAY! 800-332-1550
For information or in CA call 408-438-1550

METAGRAPHICS
SOFTWARE CORPORATION
269 Mount Hermon Road
Scotts Valley, CA 95066

Number 143 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CUT PROGRAMMING TIME. WITH GRAPHIC RESULTS.



Get to the market quicker with graphics that don't compromise performance. GSS® Graphics Development Toolkits for DOS and OS/2 give you access to more than 100 high-level graphics functions. And your GDT-based applications will support over 300 graphics devices.

The GDT is a time-proven solution for programs written in C, FORTRAN, Pascal, BASIC Compiler or Macro Assembler. It has been optimized through eight years of refinement and application to produce high-quality, high-performance graphics.

YOU'LL BE IN GOOD COMPANY.

IBM has licensed the GDT for its three PC operating systems: DOS, OS/2 and AIX. SPC's Harvard Graphics and Ashton-Tate's Draw APPLAUSE use GDT technology. So do hundreds of other PC packages for science, business and engineering.

IN A HURRY?

CALL (503) 641-2455.

ASK FOR DEPARTMENT PJ

Order your GDT, or ask for a free copy of our Devices Supported booklet. The GDT is the most direct route to completing your applications—with a graphic difference.

The GSS logo and GSS are registered trademarks of Graphic Software Systems, Inc.

Graphic Software Systems

Number **285** on the Reader Service Card

34010 Video Co-Processors from PC Tech

- Mono 2048 x 1536 displayed pixels
- Color 1024 x 768 displayed pixels
- Up to 5 M byte memory
- Optional CCITT imaging, TIGA, & DGIS software drivers
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- PC/XT/AT compatible
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PTech

907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, and TIGA are trademarks or registered trademarks of their respective companies.

Number **208** on the Reader Service Card

Celebrating our 10th Anniversary!

AMX™ 86

KADAK's engineers bring years of practical real-time experience to over 600 installations world-wide.

This *real time* MULTITASKING KERNEL simplifies *real life* product development

- No royalties
- IBM PC DOS® support
- C, PL/M, Pascal
- Preemptive scheduler
- Time slicing if needed
- Essential source code included
- Intertask messages
- Dynamic operations
  - task create/delete
  - task priorities
  - memory allocation
- Event Manager
- Semaphore Manager

AMX86™ v2.0 for 8086/88, 80186/88, 80286 systems

Demo package	**$25 US**
Manual only	**$75 US**
AMX86 system	**$2195 US**
Full source Add	**$805 US**

(Shipping/handling extra)

Also available for 8080, Z80, 68000

KADAK Products Ltd.
206 - 1847 W. Broadway
Vancouver, B.C. Canada V6J 1Y5
Telex: 04-55670
Fax: (604) 734-8114
Telephone: (604) 734-2796

Number **211** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Measuring Performance" by Michael Abrash

1

LISTING 1

```
; ******************************************************
; * From The Zen of Assembler. Appears by permission of *
; * Scott, Foresman & Company.                          *
; ******************************************************
;
; Uses the 8253 timer to time the performance of code that takes
; less than about 54 milliseconds to execute, with a resolution
; of better than 10 microseconds.
;
; By Michael Abrash 4/14/89
;
; Externally callable routines:
;
;   TimerOn: Starts the timer, with interrupts disabled.
;
;   TimerOff: Stops the timer, saves the timer count,
;       times the overhead code, and restores interrupts to the
;       state they were in when TimerOn was called.
;
;   TimerReport: Prints the net time that passed between starting
;       and stopping the timer.
;
; Note: If more than about 54 ms passes between TimerOn and
;       TimerOff calls, the timer turns over and the count is
;       inaccurate. When this happens, an error message is displayed
;       instead of a count.
;
; Note: Interrupts *MUST* be left off between calls to TimerOn
;       and TimerOff for accurate timing and for detection of
;       timer overflow.
;
; Note: These routines can introduce slight inaccuracies into the
;       system clock count for each code section timed even if
;       timer 0 doesn't overflow. If timer 0 does overflow, the
;       system clock can become slow by virtually any amount of
;       time, since the system clock can't advance while the
;       timer is timing. Consequently, it's a good idea to reboot
;       at the end of a timing session. (The battery-backed clock,
;       if any, is not affected.)
;
; All registers, and all flags except the interrupt flag, are
; preserved by all routines. Interrupts are enabled and then disabled
; by TimerOn, and are restored by TimerOff to the state they were
; in when TimerOn was called.
;

Code    segment word public 'CODE'
        assume  cs:Code, ds:nothing
        public  TimerOn, TimerOff, TimerReport

;
; Base address of the 8253 timer chip.
;
BASE_8253               equ     40h
;
; The address of the timer 0 count registers in the 8253.
;
TIMER_0_8253            equ     BASE_8253 + 0
;
; The address of the mode register in the 8253.
;
MODE_8253               equ     BASE_8253 + 3
;
; The address of Operation Command Word 3 in the 8259 Programmable
; Interrupt Controller (PIC) (write only, and writable only when
```

2

```
; bit 4 of the byte written to this address is 0 and bit 3 is 1).
;
OCW3                    equ     20h
;
; The address of the Interrupt Request register in the 8259 PIC
; (read only, and readable only when bit 1 of OCW3 = 1 and bit 0
; of OCW3 = 0).
;
IRR                     equ     20h
;
; Macro to emulate a POPF instruction while fixing the bug in some
; 80286 chips which allows interrupts to occur during a POPF even when
; interrupts remain disabled.
;
MPOPF macro
        local   p1, p2
        jmp short p2
p1:     iret                    ;jump to pushed address & pop flags
p2:     push    cs              ;construct far return address to
        call    p1              ; the next instruction
        endm

;
; Macro to delay briefly to ensure that enough time has elapsed
; between successive I/O accesses so that the device being accessed
; can respond to both. Jumping flushes the prefetch queue, forcing
; a memory access.
;
DELAY   macro
        jmp     $+2
        jmp     $+2
        jmp     $+2
        endm

OriginalFlags           dw      ?       ;processor flags in effect
                                        ; when TimerOn called
TimedCount              dw      ?       ;timer 0 count when the timer
                                        ; is stopped
ReferenceCount          dw      ?       ;number of counts required to
                                        ; execute timer overhead code
OverflowFlag            db      ?       ;used to indicate whether the
                                        ; timer overflowed during the
                                        ; timing interval
;
; String printed to report results.
;
OutputStr       label   byte
                db      0dh, 0ah, 'Timed count: ', 5 dup (?)
ASCIICountEnd   label   byte
                db      ' microseconds', 0dh, 0ah
                db      '$'
;
; String printed to report timer overflow.
;
OverflowStr     label   byte
        db      0dh, 0ah
        db      '*****************************************************'
        db      0dh, 0ah
        db      '* The timer overflowed; the interval timed was too *'
        db      0dh, 0ah
        db      '* long for the timer to measure.                   *'
        db      0dh, 0ah
        db      '*****************************************************'
        db      0dh, 0ah
        db      '$'

;*********************************************************************
;* Routine called to start timing.                                   *
;*********************************************************************

TimerOn         proc    near

;
; Save the context of the program being timed.
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code, continued

3

```
;
        push    ax
        pushf
        pop     ax                      ;get flags so we can force
                                        ; interrupts off when leaving
                                        ; this routine
        mov     cs:[OriginalFlags],ax
        and     ax,0fdffh               ;set pushed interrupt flag
                                        ; to 0
        push    ax
;
; Turn on interrupts, so the timer interrupt can occur if it's
; pending.
;
        sti
;
; Set timer 0 of the 8253 to mode 2 (divide-by-N), to cause
; linear counting rather than count-by-two counting. Also
; leaves the 8253 waiting for the initial timer 0 count to
; be loaded.
;
        mov     al,00110100b            ;mode 2
        out     MODE_8253,al
;
; Set the timer count to 0, so we know we won't get another
; timer interrupt right away.
; Note: this introduces an inaccuracy of up to 54 ms in the system
; clock count each time it is executed.
;
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;lsb
        DELAY
        out     TIMER_0_8253,al         ;msb
;
; Wait before clearing interrupts to allow the interrupt generated
; when switching from mode 3 to mode 2 to be recognized. This delay
; must be at least 210 ns long to allow time for that interrupt to
; occur. Here, 10 jumps are used for the delay to ensure that the
; delay time will be more than long enough even on fast 80386
; computers.
;
        rept 10
        jmp     $+2
        endm
;
; Disable interrupts to get an accurate count.
;
        cli
;
; Set the timer count to 0 again to start the timing interval.
;
        mov     al,00110100b            ;set up to load initial
        out     MODE_8253,al            ; timer count
        DELAY
        sub     al,al
        out     TIMER_0_8253,al         ;load count lsb
        DELAY
        out     TIMER_0_8253,al         ;load count msb
;
; Restore the context and return.
;
        MPOPF                           ;keeps interrupts off
        pop     ax
        ret

TimerOn         endp

;*************************************************************************
;* Routine called to stop timing and get count.                          *
;*************************************************************************

TimerOff proc   near
```

4

```
;
; Save the context of the program being timed.
;
        push    ax
        push    cx
        pushf
;
; Latch the count.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
;
; See if the timer has overflowed by checking the 8259 for a pending
; timer interrupt.
;
        mov     al,00001010b            ;OCW3, set up to read
        out     OCW3,al                 ; Interrupt Request register
        DELAY
        in      al,IRR                  ;read Interrupt Request
                                        ; register
        and     al,1                    ;set AL to 1 if IRQ0 (the
                                        ; timer interrupt) is pending
        mov     cs:[OverflowFlag],al    ;yes, it did indeed overflow
;
; Allow interrupts to happen again.
;
        sti
;
; Read out the count we latched earlier.
;
        in      al,TIMER_0_8253         ;least significant byte
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;most significant byte
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to elapsed
                                        ; count
        mov     cs:[TimedCount],ax
; Time a zero-length code fragment, to get a reference for how
; much overhead this routine has. Time it 16 times and average it,
; for accuracy, rounding the result.
; Note: ReferenceTimerOn resets the timer to 0, which introduces
; an inaccuracy of up to 54 ms in the system clock count each time
; it is executed.
;
        mov     cs:[ReferenceCount],0
        mov     cx,16
        cli                             ;interrupts off to allow a
                                        ; precise reference count
RefLoop:
        call    ReferenceTimerOn
        call    ReferenceTimerOff
        loop    RefLoop
        sti
        add     cs:[ReferenceCount],8   ;total + (0.5 * 16)
        mov     cl,4
        shr     cs:[ReferenceCount],cl  ;(total) / 16 + 0.5
;
; Restore original interrupt state.
;
        pop     ax                      ;retrieve flags when called
        mov     cx,cs:[OriginalFlags]   ;get back the original flags
        and     ch,not 0fdh             ;only care about original
                                        ; interrupt flag...
        and     ah,0fdh                 ;...keep all other flags in
                                        ; their current condition
        or      ah,ch                   ;make flags word with original
                                        ; interrupt flag
        push    ax                      ;prepare flags to be popped
;
; Restore the context of the program being timed and return to it.
;
        MPOPF                           ;restore the flags with the
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

5

```
                                        ; original interrupt state
        pop     cx
        pop     ax
        ret

TimerOff endp

;
; Called by TimerOff to start timer for overhead measurements.
;

ReferenceTimerOn        proc    near
;
; Save the context of the program being timed.
;
        push    ax
        pushf           ;interrupts are already off
;
; Set timer 0 of the 8253 to mode 2 (divide-by-N), to cause
; linear counting rather than count-by-two counting.
;
        mov     al,00110100b    ;set up to load
        out     MODE_8253,al    ; initial timer count
        DELAY
;
; Set the timer count to 0.
;
        sub     al,al
        out     TIMER_0_8253,al ;load count lsb
        DELAY
        out     TIMER_0_8253,al ;load count msb
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     ax
        ret

ReferenceTimerOn        endp

;
; Called by TimerOff to stop timer and add result to ReferenceCount
; for overhead measurements.
;

ReferenceTimerOff proc near
;
; Save the context of the program being timed.
;
        push    ax
        push    cx
        pushf
;
; Latch the count and read it.
;
        mov     al,00000000b            ;latch timer 0
        out     MODE_8253,al
        DELAY
        in      al,TIMER_0_8253         ;lsb
        DELAY
        mov     ah,al
        in      al,TIMER_0_8253         ;msb
        xchg    ah,al
        neg     ax                      ;convert from countdown
                                        ; remaining to amount
                                        ; counted down
        add     cs:[ReferenceCount],ax
;
; Restore the context of the program being timed and return to it.
;
        MPOPF
        pop     cx
        pop     ax
        ret
```

6

```
ReferenceTimerOff endp

;*****************************************************************
;* Routine called to report timing results.                      *
;*****************************************************************

TimerReport     proc    near

        pushf
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    ds
;
        push    cs
        pop     ds
        assume  ds:Code
;
; Check for timer 0 overflow.
;
        cmp     [OverflowFlag],0
        jz      PrintGoodCount
        mov     dx,offset OverflowStr
        mov     ah,9
        int     21h
        jmp     short EndTimerReport
;
; Convert net count to decimal ASCII in microseconds.
;
PrintGoodCount:
        mov     ax,[TimedCount]
        sub     ax,[ReferenceCount]
        mov     si,offset ASCIICountEnd - 1
;
; Convert count to microseconds by multiplying by .8381.
;
        mov     dx,8381
        mul     dx
        mov     bx,10000
        div     bx              ;* .8381 = * 8381 / 10000
;
; Convert time in microseconds to 5 decimal ASCII digits.
;
        mov     bx,10
        mov     cx,5
CTSLoop:
        sub     dx,dx
        div     bx
        add     dl,'0'
        mov     [si],dl
        dec     si
        loop    CTSLoop
;
; Print the results.
;
        mov     ah,9
        mov     dx,offset OutputStr
        int     21h
;
EndTimerReport:
        pop     ds
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        MPOPF
        ret

TimerReport     endp
```

Code continues . . .

PRODUCED 2024 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

7

Abrash code, continued

```
Code    ends
        end

LISTING 2

;
; Program to measure performance of code that takes less than
; 54 ms to execute.
;
; Link with Listing 1.
;
stack   segment para stack 'STACK'
        db      512 dup(?)
stack   ends

Code    segment para public 'CODE'
        assume  cs:Code, ds:code
        extrn   TimerOn:near, TimerOff:near, TimerReport:near
Start   proc    near
; Start the timer.
        call    TimerOn
; ***** Code to be measured starts here *****
        rept    1000
        shr     dx,1
        shr     dx,1
        shr     dx,1
        add     ax,dx
        endm
; ***** Code to be measured ends here *****
;
; Stop the timer.
        call    TimerOff
; Display the results.
        call    TimerReport
; Terminate the program.
        mov     ah,4ch
        int     21h
Start   endp
Code    ends
        end     Start
```


The Fastest Fortran 77/386 Compiler Around!!!

Proven compilers get solid results

- Compiles 13,000 lpm on a 386
- Runs under MS-DOS with DOS extenders
- Optional checkout mode with run-time diagnostics
- Full window-based source level debugger
- Dynamically allows linking to libraries during program's execution
- Contains a full 32-bit in-line symbolic assembler with window-based debugger
- Addresses up to 4 Gigabytes of memory

OTG SYSTEMS, INC.™

Distributed by OTG Systems, Inc.

Our compilers run faster, do more, cost less and increase system and people productivity!

P. O. Box 239, Clifford, Pennsylvania 18413-0239

(717) 222-9100 Fax: (717) 222-9103

MS-DOS is a trademark of Microsoft Corporation. FTN77/386 is developed by the UNIVERSITY OF SALFORD

Number **270** on the Reader Service Card

Are you still fussing with floppies for BACKUP? Our QIC40 JUMBO tape drive from MISOSYS is your solution!

The Colorado Memory Systems' JUMBO tape drive is one drive that fits all computers. It comes ready for internal use in AT's, XT's, and PC's, and connects to your floppy disk controller. Our adaptor board can be used to connect JUMBO when all FDC connectors are in use. Kits convert Jumbo to external use.

- In about 5.5 minutes, JUMBO backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000 tape cartridges.
- JUMBO plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Our tape adapter board mounts in your host computer to provide two more floppy ports — especially useful in 286 and 386 computers. When used with our external JUMBO, it lets you share your JUMBO drive between computers.
- JUMBO has custom chips, high speed brushless motors, automatic circuits, no pots to vibrate out of calibration, and fewest parts to assure long life and technical leadership for years to come.
- By using advanced Reed-Soloman error correction, JUMBO typically backs up more than 100,000 cartridges before encountering its' first error — of less than one error per 1000 years if the cartridge is fully backed up every working day.

Jumbo	$295 ($5S&H)
Tape Adaptor	$75 ($3S&H)
External Adaptor	$110 ($5S&H)
DC2000 tape	$22.50 ($2S&H)

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
or 703-450-4181

Number **238** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
NEW!	C+O Data Structures Library (multiple inheritance, messaging, excpetion handling, call-back)	$375
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
NEW!	Code Base (database manager, dBase and Clipper compatibile indes & data files; Version 4.0)	$260
	Greenleaf Data Windows (windows, menus, data entry, interactive form design; specify compiler)	$220
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$175
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
	Sherlock (C debugging aid)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
NEW!	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	Greenleaf Functions (296 useful C functions, all DOS services; specify compiler)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	TurboGeometry (library of routines for computational geometry)	$125
Updated!	Install 2.1 (automatic installation program; user-selected partial installation; CRC checking)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
	Minix Operating System (U**x-like operating system, includes manual)	$105
	HyperText Viewer (simple hypertext system; multi-file documents; includes Tiny Curses)	$100
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, update by Dan Lanciani)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	Tele Operating System (TeleKernel, TeleWindows, TeleFile, & TeleBTree by Ken Berry)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
	evalO (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	Kinetic Image Synthesizer (3-D animation system ... Saturday morning on your PC!)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	Icon-Tools (full-featured icon display and editing system)	$65
NEW!	SYSKIT (rommable or TSR debug/monitor; easily expandaded)	$60
	Quincy (interactive C interpreter)	$60
	Symtab (general-purpose symbol table construction and management package)	$60
	PTree (general-purpose parse tree construction and management package)	$60
NEW!	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card; Version 4.1.1)	$50
	Make (macros, all languages, built-in rules)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	OOPS (collection of handy C++ classes by Keith Gorlen of NIH; Version 2.2)	$35
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor; now includes documentation)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	SP (spelling checker with dictionary and maintenance tools)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; Version 1.1)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	C/reativity (Eliza-based notetaker)	$15
	Data	
Updated!	GenBank DNA Sequences (Release 59.0; includes demo disk of Pearson FAST/A programs)	$150
	Protein Sequences (over 10,000 sequences; includes demo disk of Pearson FAST/A programs)	$60
NEW!	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	Dictionary Words (234,932 words in alphabetical order, no definitions)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
NEW!	USNO Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions)	$30
	King James Bible (Old and New Testaments)	$25
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8831
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Number 30 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Simple Terminal Program in Presentation Manager

by
William S. Hall

Increasingly often, personal computers are being used as terminals. With suitable software, a PC can emulate a variety of such devices for a much lower unit cost. In addition, using capture techniques or file-transfer protocols, data can be exchanged reliably between the host and the local machine. And although the LAN explosion is generating new ways to link PCs or PCs and other types of machines, the classic approach of simply pretending to be a terminal will continue to be an important method of information exchange.

In this article, we will show you how to transform your powerful, expensive OS/2 machine into a dumb terminal. However, by keeping it simple, we can exhibit some useful programming techniques, such as accessing the OS/2 communications interface, loading and drawing with fixed-pitch fonts, creating a thread to read the serial port, and using a dialog box to change communications parameters.

PMTERM is a "real" PM program; that is, it has its own window and uses the graphical interface to draw the text. The display code itself was discussed in two previous articles (see references). The presentation on the screen is similar to a teletypewriter: The window scrolls upward, and the next line of text is displayed at the bottom.

Since PMTERM is a character-based application, it can also be written as a so-called windowed-text application. If you are using OS/2 version 1.10, you have already seen an example of such a program, namely the OS/2 command shell. However, one can envision many interesting possibilities in terminal emulation where this kind of display would be inadequate. An obvious example is having your terminal program respond to graphics commands such as the Tektronics protocol; another is displaying nonstandard character sets or a variety of character sizes.

For example, with a minimum of equipment (a terminal unit and a good shortwave radio), you can read RTTY (radio Teletype) and Morse code with your computer. However, many of these broadcasts require characters other than roman letters to display correctly; for example, they may be in Russian, Greek, Arabic, or Japanese or include stock-report fractions or weather symbols. By loading the proper font resource (which you will probably have to create), you can see the display as it was intended to appear. Finally, in some wide-area networks, the user interface is a graphical one. So, there are many reasons to use PM to its fullest extent in your programs.

Although PMTERM is a fairly short program, it is still too long to print in the magazine; as usual, the complete source and executable codes can be found on the *PJ* code disk.

Running the Program

PMTERM is quite easy to use. Run OS/2 version 1.10 or later, and copy PMTERM.EXE and PMTERM.INI to a directory in your path. Edit PMTERM.INI to reflect your communications environment. This file consists of one line:

```
portnum,baudrate,parity,wordsize
```

where each field is an ASCII string of digits. The accepted values are

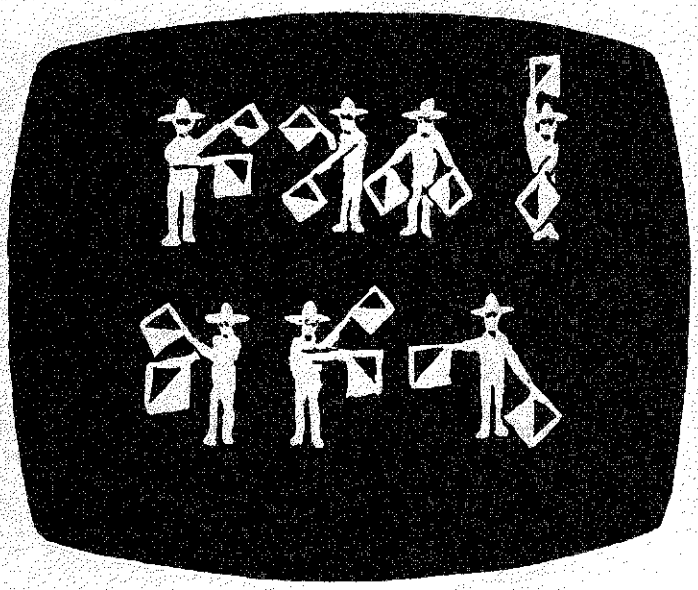
```
portnum:        1 or 2
baudrate:       as required
parity:         0 = none
                1 = odd
                2 = even
                3 = mark
                4 = space
wordsize        7 or 8
```

In this program, the number of stop bits is managed for you. If the baud rate is greater than 110, then one stop bit is set; otherwise, two are used.

The range of acceptable baud rates deserves some comment. OS/2 documentation states that only the values of 110, 150, 300, 600, 1200, 2400, 4800, 9600, and 19,200 are accepted. In fact, rates at least as low as 45 baud can be set, and you can try any value in the range 45 to 19,200. You may not always be successful, but Microsoft apparently has made an attempt to admit as many values as possible by approximating some rates by nearby values so long as the error does not exceed international standards. For example, you cannot set 134.5 baud, but 136 is accepted. (If you are having a hard time imagining such rates as being of interest, RTTY stations, for example, transmit at speeds from 45.5 to 74.2 baud using 5-bit Baudot code. There are thousands of such broadcasts around the world supporting the military, diplomatic corps, press, amateur radio, weather stations, maritime interests, clandestine operations . . .)

If the INI file is not found, terminal parameters default to 1200 baud, 8 bits, no parity, and communications port 1. If either port 1 or 2 is requested but is not available, the program is smart enough to try the other one. Note also that the present code does not allow for

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



COM3, which is available on some OS/2 machines. You should change the code in the places noted in the source if you need this capability (I did not add it because I could not test its operation).

If the initialization fails, the program exits, perhaps without any messages. However, you will get some indication of what is wrong if some of the communications parameters cannot be set. The sparse use of informative error messages is a weakness of PMTERM. Another shortcoming is that no smart terminal emulation is provided. In fact, when you consider how the text is displayed, you realize that a major modification of this section will be necessary to add some intelligence to the program. Sometime in the future, I will try to provide an example of a DEC VT52 emulator and give you a basis for building a more elaborate communications program.

When the program is running, you can make temporary changes to the port in use, baud rate, parity, and word size using a dialog box. The only other option is to display an "about" box.

Comparisons with Windows

In a future issue, I am going to show you how to write the same program in Microsoft Windows. As you would expect, there are many similarities. For example, the user interfaces in the two are essentially the same; if you can use the PM version, you will have no surprises when you try the program in Windows, and vice versa. The dialog boxes contain similar selections, and the text displays from the bottom up in both. (In fact, this part of the code is shared between the PM and Windows versions.)

From the programmer's viewpoint, there are some significant differences, however. First, there is the problem of changing the code writer's mindset from Windows to PM. Since there are *no* functions with the same names (although many are similar) the problem is analogous to switching among assembly languages for different processors. Charles Petzold, in an interview for *PC WEEK*, remarked on this problem. I have to agree with him that it is difficult.

Here are some other differences. Dialog box design is simpler (for me) in Windows than in PM. First, the coordinate system in the two differ in how the vertical direction is referenced. As I tend to build the box from the top down, I have more difficulties in PM than in Windows. Another complication is that the default font in PM, which is used in menus, dialog boxes, and title bars, is proportional. Hence it is sometimes difficult to calculate spacing in and between objects.

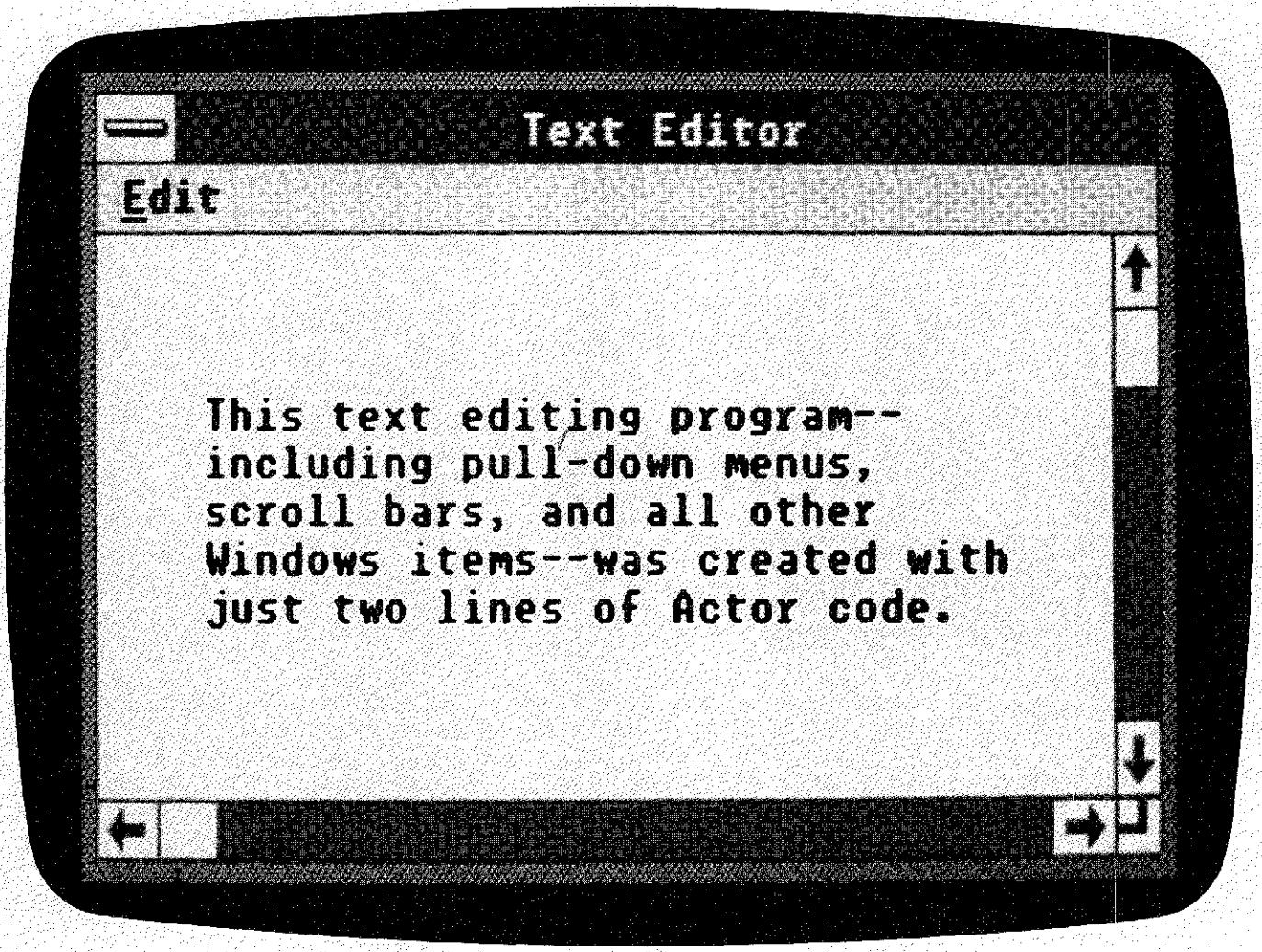
In a character-based program such as PMTERM, it is much more convenient to use a fixed-pitch font for display. Hence, it is necessary to enumerate the fonts, find one of suitable size, and then use it in the program. In contrast, the default font in Windows is fixed pitch, and many of the subsequent complications in this area simply disappear.

Although multitasking, Windows has no threads that can be invoked in a program. In a communications program, it is necessary to poll the serial port, read the contents into a buffer if any data is present, and dispatch the buffer to the window responsible for the display. Of course, the program cannot wait if the port has no data, and, since this is Windows, messages must continue to be read so that other programs can also run. So, the **PeekMessage** technique is needed to retrieve and process any messages, and, if there are none, poll the communications port. Although the method is effective, it is not very elegant, but it does provide the correct synchronization between displaying incoming data and message processing. (If you have read my articles in *PJ* 7.1 and 7.3, then you are already familiar with the method.)

In PM, a thread can be created to handle reading the communications port. This is a separate process that calls the serial port device driver. If there are any characters read, they are dispatched in a buffer to the display window via **WinPostMsg** using a user-defined message. Of course, since the buffer is shared between two threads, its use must be synchronized with a semaphore. Hence, in the PM version, **PeekMessage** is not required, and the message loop is simpler. The additional complication of the semaphore is not great.

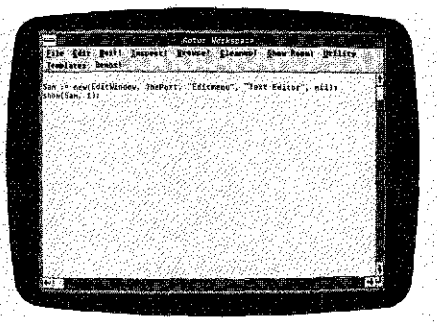
In PM, the communications interface is installed as part of the operating system as an OS/2 device driver. Thus, the port is treated as a file when it is opened, read, written, and closed, and device modifications are accomplished by IOCtl calls. In Windows, the communications handler is a dynamic link library (DLL); hence, it has its own set of function calls to perform its operations. In both cases the interface is rich enough to perform most operations,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Actor™ is the fastest, easiest way to develop applications for Microsoft® Windows. The reason—Actor's reusable toolkit of objects such as dialog boxes and edit windows. It more than doubles your overall productivity, making Actor an essential part of any Windows development project.

A full-featured, interactive, Windows-based programming environment, Actor provides immediate compilation, interactive testing, and source-code debugging. You can use it to produce fast standalone applications that support all Windows features, including Dynamic Data Exchange (DDE) and expanded memory.



These two lines of Actor code are all it took to produce the Windows text editing program you see above. Just think what you can do with a few more lines.

Actor is a powerful, pure object-oriented programming language. It's all you need to develop complete Windows applications. You also have the option to dynamically link Microsoft C code to your Actor program.

Either way, it's the fastest way to produce everything from prototypes to complete development projects. No wonder so many developers are already using Actor.

Call us now for more information.

The sooner you do, the sooner you can speed up *your* Windows development work.

Find out about Actor.
The Whitewater Group, (312) 491-2370

Technology Innovation Center
906 University Place, Evanston, IL 60201

© Copyright 1989, The Whitewater Group

Number **73** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and the results are similar. One missing item that might be useful in more unusual applications of the communications device is the ability to read the state of the control lines (DSR, for example). An event mask can be set to indicate when the state has changed, but so far as I can discern, the new state itself cannot be ascertained. Another useful feature would be the ability to set the port to a loopback mode. In this way, terminal local echo and terminal offline operation could be more accurately rendered in OS/2 or Windows.

Despite a lot of effort by Microsoft, the state of the documentation of the PM Software Development Kit as supplied to subscribers is far behind that of Windows. Probably the worst shortcoming is the lack of *printed* reference material for PM. The on-line help in the QuickHelp program is good, but it is very difficult to study the material except by reading it on the screen. You can, of course, print items of interest, but it is just not the same as having the information in a book that you can read in places other than in front of the computer (for example, in the hot tub or at the beach). PM is a very complicated system, and it requires much study. Worst of all, there is little information about the overall architecture of Presentation Manager, and no overview of related functions is available. There have been promises, but as yet we are still working with early manuals that are hopelessly out of date. Of course, if you do not mind spending even more money, you *can* purchase a rather complete package of documentation from IBM. It is a pity that Microsoft cannot supply something of similar quality to its paying customers at this time.

Structure of PMTERM

PMTERM consists of four compilation modules in three code segments and is created in the medium memory model. The window procedure, supporting functions, and display code are in a resident segment marked **PRELOAD** and **MOVEABLE**. All initialization code is in a separate segment marked **PRELOAD**, **MOVEABLE**, and **DISCARDABLE**. Infrequently used items such as the "about" and "settings" dialog box functions and supporting code are in a separate segment marked as **LOADONCALL**, **MOVEABLE**, and **DISCARDABLE**. The initial *HEAPSIZE* is 4K, and the initial *STACKSIZE* is 8K. (I found that a 4K stack was not adequate to handle the font enumeration call during initialization.) Finally, three functions are exported: the window procedure and the two dialog box functions.

The Window Procedure and the Read Comm Thread

Listing 1 shows the elements of the window procedure and the thread that reads the communications port and dispatches the buffer to be displayed. For brevity, most of the actual code has not been shown; however, the fairly detailed set of comments should make clear the actions taken for each message.

The thread is simplicity itself. Note in particular how the semaphore is used to signal the thread that the buffer has been processed and that it is safe to attempt to fill it again by reading more characters into it from the port. Note also that all waiting for the semaphore to clear is done in the thread: You do not want to wait while processing a message, as all other PM programs will stop running as well.

Learning how to process **WM_CHAR** caused me great difficulty. When this code was first attempted, the current version of the SDK was 1.05. It contained an interesting bug: After running the compatibility box, upon return to OS/2, all shifted keys returned unshifted character codes. Now, during that time, I was using the compatibility box very frequently because most of my tools were MS-DOS programs. Naturally, no matter how I juggled the key flag and the character values, the results were never satisfactory. Later, when version 1.06 appeared, the bug disappeared, but the workarounds I had developed had to be scrapped. Eventually, by using PMAUX (see *PJ* 7.2), I was able to determine the actions that transpired when a key was pressed and released. The resulting code seems to be reasonable when processing 7-bit ASCII characters; I cannot guarantee its robustness otherwise.

Program Initialization

Listing 2, using a combination of text and code, shows the initialization sequence. Two areas of particular interest are the steps for opening and setting the serial port and finding a suitable fixed-pitch font.

All PM programs begin with a call to **WinInitialize**, which must be the first call by any thread using PM facilities. The argument is a 16-bit parameter, but in PM, only NULL is allowed. It means that the initial state for newly created windows has all messages available for processing. The return value from the function is an anchor block handle, but interestingly enough, it is not really needed for calls that use it, since OS/2 deduces its value from the identity of the calling thread. The main requirement for obtaining and using the handle in those calls where it is a parameter is for upward compatibility with IBM's SAA architecture. Of course, if NULL is returned for the handle, then the program cannot continue.

The anchor block handle is then passed along with the requested queue size to **WinCreateMsgQueue**, the very next function that must be called. A queue-size parameter of zero signals the system to use the default size. A return value of NULL indicates an error, which can be determined from **WinGetLastError**. In fact, errors in many PM functions can be analyzed by using this function.

In a previous article, I noted that I had learned how to use some of the local heap functions by reading through several sample programs until I found a suitable example. Now I can explain the six zeros used when calling **WinCreateHeap** to get a handle to the heap. Parameter one, the heap base, is a selector of the segment to contain the local heap. Parameter two is the heap size. A value of 0 for both means that the heap will be located at the end of the program's automatic data segment and the size will be the value specified in the *HEAPSIZE* keyword in the definitions file.

Parameter 3 is the heap growth size, in bytes. A value of zero means that the heap will grow a minimum of 512 bytes if its size must be increased to satisfy a request for memory.

Parameters 4 and 5 are the minimum and maximum element sizes for a dedicated free list. One dedicated free list is created for each element size (in multiples of four bytes) to cover the range of element sizes specified. If the values of both are zero, then no element size has a dedicated free list and all sizes are on the nondedicated free list. The benefit of having dedicated free lists is speed in allocation, at the cost of two extra bytes for each list in the heap control block.

The last parameter can contain two flag bits. One specifies that the created

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

heap should support moveable objects, and the other indicates that the size parameter will be checked in **WinFreeMem**. A value of zero indicates that the allocated area remains at a fixed location in the heap.

If **WinCreateHeap** is successful, the handle becomes a parameter to all subsequent calls to the heap manager functions. There are also a number of cautions that must be exercised if calls are made to other OS/2 or C-language memory-allocation functions, since there may be conflict. In this case, **DosAllocSeg** or **DosAllocShrSeg** must be called to retrieve a segment selector, which is then used to create the heap in a different segment. Finally, the *argv* value (or values) should be saved before the call to create the heap.

The register window class function permits allocating extra bytes of data per window created for application use. Although it is not used in PMTERM, many uses of this parameter can be envisioned. One that I have found very helpful is storing a pointer to a unique global data structure associated with a particular window. This pointer can be recovered very easily using **WinQueryWindowUShort** if the window handle is known. In this way, a number of windows of the same class can be supported by the same code.

WinCreateStdWindow has had many incarnations as PM has developed, resulting in incompatibility from one version to the next. The principal problem has been in the meaning of the frame-creation parameter. Currently it is a long, and the create function receives its address. Its value is a combination of the FCF_* flags, and if it is not precisely rendered, the window will fail to display. If you are having difficulty in running programs written for earlier versions of PM, the culprit is probably here.

The best way to assign this parameter is to begin with the value *FCF_STANDARD* and then mask out those properties *not* possessed by your window. For example, the window in PMTERM has neither an icon nor an accelerator table. Hence, the value should be *FCF_STANDARD & ~FCF_ACCELTABLE & ~FCF_ICON*. In some early programs I built up the value by simply ORing the appropriate attributes. However, *FCF_STANDARD* apparently contains bits that are essential for the function to work, and the proper technique now is to remove what you do not need.

Another tricky parameter is the resource ID. Assuming all resources are in the executable's resource segment (which is the case if the previous parameter is NULL), it is important that the menu, accelerator table, and icon have the same identifier. And if another window of the same type is to be created to use these resources, then it may be necessary to change the previous window's value using **WinSetWindowUShort**. PM is easy to program—if you can just remember everything.

Creating a window generates a **WM_CREATE** message. In PMTERM, **WndCreate** is then called where a suitable font is found, and the text buffer for the screen is created. If either fails (as indicated by either a NULL frame window handle or a NULL pointer to the buffer), the program must exit. If successful, PMTERM.INI is read and the values obtained are used to set the serial port. If all is well, the window is shown.

The following steps are required for using the communications port:

1. Open the port and obtain a handle.
2. Set the baud rate.
3. Set parity, word size, and stop bits.
4. Set the device control block.

Later, we will examine the functions that perform these operations.

Loading and Drawing Text with a Fixed-Pitch Font

The techniques described here actually have a broader applicability than to fixed-pitch fonts. But limiting the discussion to this situation, here are the steps. You can follow along by reading the **WndCreate** code at the end of Listing 2. The method used is rather heavy-handed; in a future article, I will try to make this section more elegant and reduce the uses of magic numbers and hard-wired file names.

1. Retrieve a cache micro presentation space handle, which is needed for the subsequent **Gpi** function calls.
2. Load the Courier font file with **GpiLoadFonts**.
3. Select the number of fonts to be enumerated; 30 seems adequate. Load the **FONTMETRICS** structure for all of them with a call to **GpiQueryFonts**. Obviously, such an operation uses a lot of stack space, since the array *fm[]* of **FONTMETRIC** structures is an automatic variable.
4. Loop through the array of structures until a font is found that is fixed pitch and has a width of 8.
5. If one is found, fill out the **FATTRS** structure, *ttyfat*, with the corresponding elements from the successful **FONTMETRIC** information, and select the desired code page. Also choose a parameter value, *fcid*, in the range 1–255 (set to 1 in this case). You will see how they are used below.
6. If no font is found having the desired characteristics, set *fcid* = 0 and get the metrics for the system font. I have assumed that the system font will *always* be the one enumerated first; I have only experimental evidence for this belief at this time.
7. Release the micro presentation space.

Listing 3 shows the code to display the text. The steps are:

1. Get a micro presentation space handle hPS.
2. If *fcid* is not zero, then create a logical font using hPS, an appropriate identifier string, *fcid*, and the **FATTRS** structure *ttyfat*.
3. Select the presentation-space character set as the just-created font.
4. Draw the text with the appropriate background mix.
5. Restore the presentation space and release it.

Finally, upon exiting the program, release the font resource as follows (see Listing 3):

1. Get a micro presentation space handle.
2. Delete the set ID associated with *fcid*.
3. Unload the fonts.
4. Release the presentation space.

The Communications Dialog Box

PMTERM has provisions for changing some of the basic communications parameters from a dialog box. The controls are quite simple. An edit box is provided for entering the new baud rate. Automatic radio buttons (ARBs) are available for setting parity (none, odd, even, mark, or space), port (1 or 2), and word size (7 or 8 bits).

Radio buttons work like the push buttons on automobile radios. At any time, one is depressed (highlighted) and the others are off. In Windows, it is necessary to manage the state of each button by turning on the one depressed and turning off the others. Later, when they must be read, the state of each must

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW BBS

SCREEN MANAGER
P·R·O·F·E·S·S·I·O·N·A·L

ADVANCED WINDOWING

• The Screen Manager Professional gives your applications the finest bullet proof windowing interface available. Although other products may boast of hundreds of functions, they often involve needless repetition. Our functions are elegant and powerful. Each accomplishes a variety of tasks with one command. All window functions are written in hand-optimized assembly language to provide the fastest possible response and smallest possible size.

ORDER

•30 day Money Back Guarantee
•Screen Manager Professional $140
•Source Code $140
•Sampler Library ..... $ 40
•Specify Compiler when ordering •Include UPS shipping: ground $5, overnight $20, 2nd day air $10, outside U.S. $30, COD $5. •Visa, Mastercard, check or money order. •Payment must be in U.S. dollars drawn on a U.S. bank. •PA residents add 6% sales tax.

POWERFUL

• Amazing Speed
• Incredible Menus
• Context Sensitive Help Editor
• OS and Compiler Independent
• Window Functions are 100% Assembly
• Keyboard Filtering for Data Entry Systems
• Full Video Adapter Support
• Compact Linkable Library
• Background Processing
• Customizable Memory Allocation Function

VERSIONS AVAILABLE FOR

•Microsoft C •Zortech C++
•Quick C •Turbo C •QuickBasic
•Turbo Pascal (available 7/89)

AMAZING

"...an amazingly complete collection of window commands." The support functions are easy to use and very powerful. This library warrants serious consideration."
Computer Language-May '89

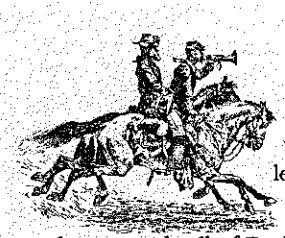
• Free yourself to concentrate on the substance of your program. Over 120 functions provide all the windowing tools needed to develop serious applications whether you are a professional or a hobbyist.
• The Screen Manager Professional can help you create easy to use, visually stimulating, programs with minimum effort. Now powerful windowing is available to you at an affordable price.

NEW BBS

Demos, Software, Tech Support- A forum for the Professional Programmer
BBS (814) 234-1881

LAI Logical Alternatives Incorporated
Calder Square, P.O. Box 10674
State College, PA 16805-0674
(814) 234-8088

Number **246** on the Reader Service Card



Turbo Taskforce.

Want to write better code in less time? We're the Turbo User Group, and since 1984 we've been helping people do that and more with all of Borland's language products. Membership in TUG will net you –

- A subscription to TUG Lines, the group's bi-monthly journal, containing utilities, bug reports, patches, workarounds, product reviews and more
- Access to TUG's "Troubleshooters" – volunteers who help solve programming problems by phone
- A discount on The TUG Boat, the group's multi-user BBS, where you can contact, correspond and conference with fellow users, participate in language SIGs and download "hot" Public Domain software
- Access to TUG's extensive disk library of hand-selected and tested Public Domain and ShareWare utilities, routines and applications
- And more!

Don't just work *harder* – join with us in our commitment to work *smarter!*

Turbo User Group
Post Office Box 1510 • Poulsbo, Washington 98370
(206) 779-9508

Annual dues (US funds): U.S. $24.00; Canada $28.00; Overseas $39.00

Number **259** on the Reader Service Card

Introducing C talk / Views™

Reduce C Programming for MS Windows by 75%

Introducing C_talk/Views™, a major productivity tool for Microsoft® Windows program development in C. Slash your MS Windows program development time by up to 75% through the powerfull object-oriented programming (OOP) technology of C_talk. With C_talk/Views, the most common elements of all MS Windows programs are packaged into reusable software components-object classes. Over forty software components are included as a complete applications development framework. These object classes provide an easy-to-learn and powerfully consistent interface to the rich functionality of MS Windows. C_talk/Views is also a real OOP environment. Use the software components as they are, or extend them to meet the most demanding applications.

Coming soon...C_talk/Views for the Apple Macintosh®... now your MS Windows program will port to the Mac with little or no change!

$450.00 In the US, add $5 for shipping. ($25 for int'l handling)

CNS, Inc.
Software Products
7090 Shady Oak Rd.
Minneapolis, MN 55344
612-944-0170, Fax 612-944-0923

. . . providing and advancing object-oriented methodology.

Contact this advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* VersiMAKE™

VersiMAKE™ is a full-featured MAKE utility that includes:

- **Dependency Generation**
  Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!
- **Wild Card File Name Matching**
  Analyzes an entire collection of source files with a single statement.
- **Nested Include Files**
  Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.
- **Powerful Macro Facilities**
  Built-in macros, user-defined macros, and environment variables.
- **Analytical Reports**
  Shows the entire Include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* VersiCREF™

VersiCREF™ is a unique utility that creates a Master Cross-Reference of your entire system.

- **Multi-Lingual**
  Handles C, assembler, or both.
- **Flexible**
  File names with line numbers, or file names alone. Global and local symbols, or globals alone.
- **Powerful**
  Easily handles systems containing 100 source files or more.

VersiMAKE™	$125
VersiCREF™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

800-344-4096
(In NJ, 609-779-0784)
MC/VISA/AMEX accepted

SUMMIT INFORMATION SYSTEMS, INC.
496 N. Kings HWY Suite 130
Cherry Hill, NJ 08034

Number 158 on the Reader Service Card

be examined in turn until the active one is found.

ARBs, new to PM, are much easier to use. The first step is to associate them in the dialog box template using the **WS_GROUP** statement. When the dialog box is displayed, touching one will automatically turn off the others in that group. Later, when the button states are to be read, sending the message **BM_QUERYCHECKINDEX** to the first button in the group will return the index of the highlighted button.

Listing 4 shows most of the code for the dialog box manager as well as the serial port supporting functions. The dialog box code itself is fairly simple. When the box is initialized (with **WM_INITDLG**), the thread reading the communications port is suspended and the current port parameters, baud rate, parity, and so on are used to initialize the state of the controls. If the action is canceled, the thread is resumed and the box is destroyed. Finally, if the OK button is depressed, the state of the buttons and the baud rate are copied to temporary variables, and an attempt is made to set the serial port accordingly.

If anything fails, a message box appears explaining the error, and that parameter does not change. However, the dialog box exits anyway. It would be friendlier to try to reset the parameters and let the user try again. (An exercise for the reader?)

Support Functions

Although we have tried to make it look easy, the serial port device driver offers many options. If you print the Quick-Help text for the **DCBINFO** structure alone, you get thirteen pages of very dense text. I will describe only our particular settings.

The first step in using the serial port is to open the device. In the function **opencomm** (Listing 4), **DosOpen** accomplishes the task. As in the rest of PMTERM, only COM1 and COM2 are allowed as device names. The device is opened as a normal file (parameter 5), and the action taken if it does not exist (parameter 6) is to open it anyway. Finally, the device is opened in read-write, non-shared mode (parameter 7).

Setting the baud is accomplished with an IOCtl call. The only parameter choices are the rate and the device handle. Another IOCtl is used to set the parity, word size, and stop bits. Both are straightforward.

The most complex action is setting the device control block (see the function **setdcb** in Listing 4). The recommended procedure is to retrieve the **DCBINFO** structure, alter the required bits, and then set it. There are nine fields in **DCBINFO**. Two are used to set the XON and XOFF characters, which default to the usual Ctrl-Q and Ctrl-S whenever the device is opened. For this program, these values are adequate. Two fields control read and write timeout values. We set the read timeout to ten seconds. Two fields determine the error and break replacement characters. We do not use them here.

Three flags control the behavior of the port under a variety of circumstances. You should note that the bit fields not specified for the flags are reserved. So, the proper way to set the bits is to mask out the ones to change and then OR them in as needed. This may explain some of the strange magic numbers in the code.

Flag 1 sets the handshaking mode. The default is to enable CTS and DSR modes. I have found a lot of problems using this mode with connector cables that do not have these lines available. The symptom is that a character is not transmitted and the program appears to lock, although reception is possible. In PMTERM, all the control bits are disabled and the DTR control mode is set. You may want to experiment with this setting. I have found this mode adequate in a variety of environments, but you may have to change it to suit your configuration and recompile. For your first attempt, try not modifying this flag at all.

The second flag enables flow-control and replacement-character modes. In PMTERM, we simply enable automatic transmit-and-receive flow control using XON/XOFF. In this way, the device driver does all the work for us.

Flag 3 enables read timeout processing. PMTERM enables a mode that operates roughly as follows: If an attempt is made to read the input data and none is available, the call will not return until the timeout value has expired; Otherwise, it returns with the characters read. In PMTERM, ten seconds is used for the timeout value. You can see the effect of this mode when trying to exit the program or changing communications ports: A delay of up to ten seconds may occur.

Why You Should Bother

The first MS-DOS machine I ever used was the Zenith Z-100. One of its nicest features was the support provided in the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

operating system for using the two serial ports, which were interrupt driver and buffered. All the programmer had to do was devise a reasonable user interface. It was much more fun to write such programs when you did not have to deal with serial port device drivers and other similar diversions taking you away from the main goal.

Today, Windows and PM provide not only the rich communications abilities so badly missed in the PC/AT class of MS-DOS machines but the user interface as well. Oh, you have to learn a new way of thinking and a few tricks as well, but they are simply a part of programming in these new environments. And now you can concentrate on doing the interesting things instead of fighting the limits of the system.

William S. Hall is a senior engineer at Olivetti Advanced Technology Center in Cupertino, California. He specializes in Windows system support software.

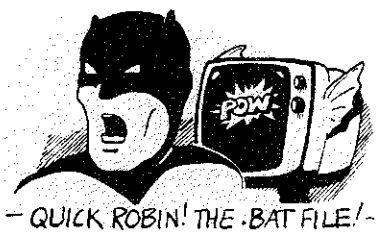
Code follows.

References

Hall, William S., "Smart Terminal Emulation in Windows," *Programmer's Journal* 7.1 (Jan/Feb 1989), 74–83.

———, "A Simple Debugging Utility for Presentation Manager and Windows," *Programmer's Journal* 7.2 (Mar/Apr 1989), 56–73.

———, "A Substitute for the Missing TYPE Command in Windows," *Programmer's Journal* 7.3 (May/Jun 1989), 64–73.



The
Heap Expander™
version 2.5

Now your programs can have virtually unlimited heap space using expanded memory, extended memory, disk space, or any combination of the three. And it's all transparent. The Heap Expander's initialization code checks the system's resources and uses whatever is available.

only $79.95*

- Uses LIM-standard expanded memory if present.
- Uses AT-style extended memory if present.
- Swaps data to disk as needed.

Libraries and Source Code for:
- Turbo C
- Microsoft C 4.0 and 5.0
- Turbo Pascal 3.0 and 4.0

Requires an IBM PC, XT, AT, or close compatible with MS-DOS or PC-DOS version 2.0 or above

MC/VISA/COD call
1-800-248-1045 x 100 (US)
1-800-952-5560 x 100 (Idaho)

*CA residents add 6 1/2% sales tax. Foreign customers, please add $4.00 for shipping and handling.

The Tool Makers
P. O. Box 2151
Santa Cruz, CA 95063
(408) 458-0690

Number **166** on the Reader Service Card

NEW VERSION 4.0

DYNAMITE DEMOS

Proteus Prototype/Demo System

Why fizzle with an outdated demo program when you can explode with Proteus?

Proteus is the first prototype/demo system to combine complete simulation capability, a powerful Screen Painter, video effects, custom sound and much more in a program you don't have to be a genius to figure out.

And now Proteus supports all graphic modes for the CGA, EGA and VGA in the popular PCX file format.

"A joy to use."
. . . Computer Language

- ***Complete Simulation Capability***
- ***Powerful Screen Painter***
- ***Bit-Mapped Graphics***
- ***Video Effects***
- ***Custom Sound***

Don't Let Your Next Demo be a Dud!
Find out why software developers from Ashton-Tate to ZyLogic have switched to Proteus.

1-800-634-9986

$149. *No royalties. 30-day Money Back Guarantee. MC/VISA/COD/PO accepted.*

Helios
P.O. BOX 22869, SEATTLE, WA 98122

Number **258** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "A Simple Terminal Program" by William S. Hall

1

```
LISTING 1
/*
 * Extracts from PMTERM's window procedure
 */

MRESULT FAR PASCAL MainWndProc(HWND hWnd, USHORT msg, MPARAM mp1, MPARAM mp2)
{

    HPS hPS;            /* handle to PM display context */
    BYTE ch;            /* keyboard character read in WM_CHAR */
    USHORT kflag;       /* keyboard flag */
    RECTL rect;         /* paint rectangle */

    switch (msg) {

        case WM_SETFOCUS:
            /* Check parameter mp2.  If non-zero then the window
               is getting the focus.  Create a flashing text cursor
               at the current insert point and show it.

               If mp2 is zero, then the window is losing the focus.
               Destroy the cursor.
            */
            break;

        case WM_SIZE:
            /*
               The window is changing size.  If we are becoming an
               icon, then we destroy the cursor and suspend the thread
               which is reading the communications port.

               Otherwise resume the thread, and create and show the cursor
               at the current insert point.
            */
            break;

        case WM_COMMAND:
            /*
               The mouse or a system key has activated a menu item.
               Only two commands are recognized:  show an 'About'
               box or display a communications parameter settings
               dialog.
            */
             break;

        case WM_USER:
            /*
               User defined message.  mp1 contains the number of
               characters read by the read comm port thread
               and placed into the global character string CommBuf.
               To display the string:
                    1.  hide the cursor;
                    2.  call TTYDisplay with the count and a pointer
                        to CommBuf;
                    3.  position the cursor at the new insert point,
                        which was updated by TTYDisplay;
                    4.  show the cursor;
                    5.  Clear the semaphore protecting CommBuf,
                        which was set by the read comm port thread.
            */
            break;

        case WM_CHAR:
            /*
               Character input. The low word of mp1 contains the
               key type flag, the high word of mp1 contains the
               auto-repeat count, and the low word of mp2 contains
               the character code.  The action here is to determine
               from the key flag and character code the proper
               value to write to the communications port.

               The steps are:
                    1.  Read the keyboard flag;
                    2.  If this is a key release take no action;
                    3.  Otherwise, read the character code.
                    4.  If the flag indicates a printing character,
                            write it to the communications port;
                    5.  If the flag indicates a control character,
                            map the character code to a control
                            character and write it to the comm port.
            */
            break;

        case WM_CREATE:
            /*
               Called when the window is created.  Here the
               fixed pitch font is chosen, the text buffer is
               created and buffer pointers initialized,
               and the read comm port thread is started.
            */
            break;

        case WM_CLOSE:
            /*
               Window is closing; perform clean up:
                    1.  suspend the read comm port thread;
                    2.  unload the font;
                    3.  post a WM_QUIT message;
            */
            break;

        case WM_DESTROY:
            /*
               Window is being destroyed.
               Force the thread to exit and close the comm device.
            */
            break;

        case WM_PAINT:
            /*
                Window requires repainting.  Retrieve a presentation
                space handle and the update rectangle and pass this
                data to the paint function.  If the window is iconic,
                then the name of the program is written to the icon
                window.  Otherwise, the characters stored in the text
                buffer corresponding to the update window are redrawn.
                When done, the presentation space is released
            */
            break;

        case WM_ERASEBACKGROUND:
            return(TRUE);

        default:        /* all other messages go here */
            return( WinDefWindowProc( hWnd, msg, mp1, mp2 ) );
            break;
    }
    return(0L);
}

/*
   This function is a separate thread of execution in the main
   program.  It is invoked during initialization and continues
   to run until the flag bExitReadThread becomes TRUE.

   As two processes use the global variable CommBuf, synchronization
   is maintained with a RAM semaphore which is set by the thread
   and cleared after the buffer is processed by the main program.

   Note that no attempt is made to handle errors.
*/
```

2

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3

```
void FAR ReadCommThread()
{
    USHORT  usNumBytes;    /* number of bytes actually read */

    while (!bExitReadThread) {
        DosSemWait(&lReadSemTrigger, -1L); /* wait for screen to be ready */
        if ((usRetCode = DosRead(hComm, CommBuf, 128, &usNumBytes)) != 0)
            ;              /* ignore errors */
        if (usNumBytes) {
            DosSemSet(&lReadSemTrigger);
            WinPostMsg(MWnd.hWnd,WM_USER,MPFROM2SHORT(0,usNumBytes),(MPARAM)0);
        }
    }
    DosExit(0,0);
}
```

LISTING 2

```
/* Extracts from the initialization segment of PMTERM */

BOOL FAR InitProgram(int argc, char *argv[])
{

  /* get anchor block handle */
    if ((hAB = WinInitialize(NULL)) == NULL)
        return FALSE;

  /* create a message queue */
    if ((hmqMsgQ = WinCreateMsgQueue(hAB,0)) == NULL)
        return FALSE;

  /* create the heap */
    if ((hHeap = WinCreateHeap(0, 0, 0, 0, 0, 0)) == NULL)
        return FALSE;

  /* register the window */
    if (!WinRegisterClass(hAB,                  /* anchor block handle */
                          (PCH)szAppName,       /* class name */
                          (PFNWP)MainWndProc,   /* window procedure for class */
                          CS_SIZEREDRAW,        /* class style */
                          0))                   /* no extra data needed */
        return FALSE;

  /* set the frame style parameter */
    ctldata = FCF_STANDARD  & ~FCF_ACCELTABLE & ~FCF_ICON;

  /* create the window */
    hwndFrame = WinCreateStdWindow(HWND_DESKTOP,        /* parent */
                                   WS_VISIBLE,
                                   &ctldata,    /* frame style */
                                   (PCH)szAppName,      /* class */
                                   (PCH)szTitle,        /* window title */
                                   0L,                  /*default client style*/
                                   (HMODULE)NULL,       /* resources in .EXE */
                                   ID_RESOURCE,         /* resource id */
                                   (HWND FAR *)&MWnd.hWnd);
                                                        /* handle to client */
    if ((hwndFrame == NULL) || (MWnd.pVidBuf == NULL))
        return FALSE;

    /* display the window */
    WinShowWindow(hwndFrame, TRUE);

    /* read initdata information */
    ReadIniFile();

    /* try to open and initialize the communications port */
    if (!InitCommPort())
        return FALSE;

    /* success */
    return TRUE;
}
```

4

```
/* called when client window is created */
void FAR WndCreate(HWND hWnd)
{

    FONTMETRICS fm[30];
    HPS hPS;
    SHORT width, height;
    SHORT cwidth, cheight;
    int i;
    LONG cFonts;
    short fcid = 0;

  /* load the icon string */
    WinLoadString(hAB, NULL, IDS_ICON, (USHORT)sizeof(szIcon), (PSZ)szIcon);

  /* get the size of an icon */
    xIconsize = WinQuerySysValue(HWND_DESKTOP, SV_CXICON);
    yIconsize = WinQuerySysValue(HWND_DESKTOP, SV_CYICON);

  /* get the system font character metrics */
    hPS = WinGetPS(hWnd);
    if (GpiLoadFonts(hAB, "c:\\os2\\dll\\COURIER.FON")) {
        cFonts = 30;
        GpiQueryFonts(hPS,3L,NULL,&cFonts,(LONG)sizeof(FONTMETRICS),fm);
        for (i = 0; i < (short)cFonts; i++) {
            cheight = (short)(fm[i].lMaxBaselineExt + fm[i].lExternalLeading);
            cwidth = (short)fm[i].lMaxCharInc;
            if ((fm[i].fsDefn & 0x8000) && (fm[i].fsType & 1) && (cwidth == 8))
                break;
        }
        if (i < (short)cFonts) {          /* we found one */
            fcid = 1L;
            ttyfat.usRecordLength  =  sizeof(FATTRS);
            ttyfat.lMatch          =  fm[i].lMatch;
            ttyfat.fsSelection     =  fm[i].fsSelection;
            strcpy(ttyfat.szFacename, fm[i].szFacename);
            ttyfat.idRegistry      =  fm[i].idRegistry;
            ttyfat.usCodePage      =  850;
            ttyfat.lMaxBaselineExt =  fm[i].lMaxBaselineExt;
            ttyfat.lAveCharWidth   =  fm[i].lAveCharWidth;
            ttyfat.fsType          =  FATTR_TYPE_FIXED;
            ttyfat.fsFontUse       =  0;
        }
    }
    if (fcid == 0L) {    /* desired font not found, use the system font */
        GpiQueryFontMetrics(hPS, (LONG)sizeof(FONTMETRICS), &fm[0]);
        cheight = (short)(fm[0].lMaxBaselineExt + fm[0].lExternalLeading);
        cwidth = (short)fm[0].lAveCharWidth;
    }
    WinReleasePS (hPS);

  /* get other parameters */
    width = (SHORT)WinQuerySysValue(HWND_DESKTOP, SV_CXFULLSCREEN);
    height = (SHORT)WinQuerySysValue(HWND_DESKTOP, SV_CYFULLSCREEN);
  /* initialize the text buffer */
    InitTTYWindow(&MWnd,0,0,width,height,cwidth,cheight,
                                                FALSE,FALSE,TRUE,fcid,0x7f);

  /* create thread to read comm port */
    bExitReadThread = FALSE;
    if ( DosCreateThread (ReadCommThread, &idThread,
                          cReadThreadStack + sizeof cReadThreadStack))
        /* this is not a very effective error response !! */
            WinAlarm(HWND_DESKTOP, WA_ERROR);
}
```

LISTING 3

```
/*
   Text display in PMTERM.  If findex is zero then the
   default (system) font is used.  If findex in
   non-zero, then the FATTRS structure ttyfat is valid and
   the font described there is used.
```

Code continues . . .

Hall code, continued

5

```
*/
  /* get presentation space */
    hPS = WinGetPS(pTTYWnd->hWnd);
    if (findex) {       /* use other than the system font */
      /* create font based on previously initialized FATTRS structure */
        GpiCreateLogFont(hPS, (PSTR8)"TTYCLS",(LONG)findex, &ttyfat);
        GpiSetCharSet(hPS,(LONG)findex);
    }
    GpiSetBackMix(hPS, BM_OVERPAINT);   /* set background */
  /* draw text */
    GpiCharStringAt(hPS,(PPOINTL)&pTTYWnd->Pos,(LONG)ctr,(PCH)str);
  /* restore the presentation space and release it */
    GpiRestorePS(hPS, -1L);
    WinReleasePS (hPS);


/*
    Steps for unloading the previously loaded font resource in PMTERM.
*/
    case WM_CLOSE:
        . . .           /* other clean-up code */

        hPS = WinGetPS(hWnd);
        GpiDeleteSetId(hPS, LCID_ALL);
        WinReleasePS(hPS);
        GpiUnloadFonts(hAB, "c:\\os2\\dll\\COURIER.FON");
        . . .
        break;
```

LISTING 4

```
/* Serial port options dialog box handler in PMTERM - extracts */

/* Dialog box to change the communications parameters */
MRESULT EXPENTRY SetParamsProc(HWND hDlg, USHORT msg, MPARAM mp1, MPARAM mp2)
{

    USHORT Baud, cid;
    BYTE Bits, Parity, Stopbits;
    HFILE hTemp;

    switch (msg) {

        case WM_INITDLG:  /* initialize controls before display */
            DosSuspendThread(idThread);
        /* set baud rate */
            WinSetDlgItemShort(hDlg, IDD_BAUD, inidata.baud, FALSE);
        /* set port radio buttons */
            WinSendDlgItemMsg(hDlg, IDD_PORT + inidata.cid - 1, BM_SETCHECK,
                                    MPFROM2SHORT(TRUE,0), 0L);
        /* set word size radio buttons */
            WinSendDlgItemMsg(hDlg, IDD_BITS + inidata.wordsize - 7, BM_SETCHECK,
                                    MPFROM2SHORT(TRUE,0), 0L);
        /* set parity radio buttons */
            WinSendDlgItemMsg(hDlg, IDD_PARITY + inidata.parity, BM_SETCHECK,
                                    MPFROM2SHORT(TRUE,0), 0L);
            break;

        case WM_COMMAND:
            switch(COMMANDMSG(&msg)->cmd) {
                case DID_OK:    /* got an OK */
                  /* read the baud rate */
                    WinQueryDlgItemShort(hDlg, IDD_BAUD, &Baud, FALSE);
                  /* read the state of the radio buttons */
                    cid = 1 + (USHORT)WinSendDlgItemMsg(hDlg, IDD_PORT,
                                            BM_QUERYCHECKINDEX, 0L, 0L);
                    Parity = (BYTE)WinSendDlgItemMsg(hDlg, IDD_PARITY,
                                            BM_QUERYCHECKINDEX, 0L, 0L);
                    Bits = (BYTE)7 + (BYTE)WinSendDlgItemMsg(hDlg, IDD_BITS,
                                            BM_QUERYCHECKINDEX, 0L, 0L);
                  /* User wants to change port.  First try opening the
                     one selected.  If successful, close the old one */
                    if (cid != inidata.cid) {
                        if (hTemp = opencomm(cid)) {
```

6

```
                            DosClose(hComm);
                            hComm = hTemp;
                            inidata.cid = cid;
                        } else  /* failed, display message */
                            ShowMessage(hDlg,IDS_ERR_OPENCOM1 + inidata.cid - 1,
                                                MB_OK | MB_ICONEXCLAMATION);
                    }
                /* try to set the baud rate, line parameters, dcb */
                    if (setbaud(hComm, Baud)) {
                        inidata.baud = Baud;
                        if (Baud <= 110)
                            Stopbits = 2;
                        else
                            Stopbits = 0;
                    }
                    else
                        ShowMessage(hDlg, IDS_ERR_SETBAUD,
                                                MB_OK | MB_ICONEXCLAMATION);
                    if (setline(hComm, Parity, Bits, Stopbits)) {
                        inidata.parity = Parity;
                        inidata.wordsize = Bits;
                        inidata.stopbits = Stopbits;
                    }
                    else
                        ShowMessage(hDlg, IDS_ERR_SETLINE,
                                                MB_OK | MB_ICONEXCLAMATION);
                    if (!setdcb(hComm))
                        ShowMessage(hDlg, IDS_ERR_SETDCB,
                                                MB_OK | MB_ICONEXCLAMATION);
                    DosResumeThread(idThread);
                    WinDismissDlg (hDlg, TRUE);
                    break;
                /* no action by user */
                case DID_CANCEL:
                    DosResumeThread(idThread);
                    WinDismissDlg (hDlg, FALSE);
                    break;

                default:
                    break;
            }
            break;

        default:
            return WinDefDlgProc(hDlg, msg, mp1, mp2);
    }
    return FALSE;
}

/* Open serial port 1 or 2 and return a file handle */
HFILE opencomm(USHORT portnum)
{

    USHORT usAction;
    char *commstr;
    HFILE hTemp;

    switch(portnum) {
        case 1:
            commstr = "COM1";
            break;
        case 2:
            commstr = "COM2";
            break;
        default:
            return (HFILE)NULL;
    }
    if (usRetCode = DosOpen(commstr, &hTemp, &usAction,0L,0,1,0x12,0L))
        return (HFILE)NULL;

    return hTemp;
}

/* set the communications baud rate */
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

7

```
BOOL setbaud(HFILE hcomm, USHORT rate)
{
    USHORT usRetCode;

    if ((usRetCode = DosDevIOCtl(0L, &rate, 0x41,1, hcomm)) != 0)
        return FALSE;
    return TRUE;
}

/* set the communications line characteristics */
BOOL setline(HFILE hcomm, BYTE parity, BYTE wordsize, BYTE stopbits)
{
    USHORT usRetCode;
    LINECONTROL slineChar;

    slineChar.bParity   = parity;
    slineChar.bDataBits = wordsize;
    slineChar.bStopBits = stopbits;

    if (usRetCode = DosDevIOCtl(0L, &slineChar, 0x42,1,hcomm))
        return FALSE;
    return TRUE;
}
/* set the communications device control block */
BOOL setdcb(HFILE hcomm)
{
    USHORT usRetCode;
    DCBINFO sDCB;

    if ((usRetCode = DosDevIOCtl(&sDCB, 0L, 0x73, 1, hcomm)) == 0) {
        sDCB.fbCtlHndShake &= 0x84;  /* mask out all control bits */
        sDCB.fbCtlHndShake |= 0x01;  /* set DTR control mode */
        sDCB.fbFlowReplace |= 0x03;  /* enable auto Xmit and recv flow control */
        sDCB.fbTimeout &= 0xf9;      /* clear read timeout flags */
        sDCB.fbTimeout |= 0x04;      /* set wait for something read timeout */
        sDCB.usReadTimeout = READTIMEOUT;       /* set read timeout value */
        if ((usRetCode = DosDevIOCtl(0L, (char *) &sDCB, 0x53,1,hComm)) != 0)
            return FALSE;
        return TRUE;
    }
    return FALSE;
}
```


4c™ A *ZOOM* key for your editor!

Hypertext editing for Programmers

Explore source code. Navigate uncharted logic trails easily!

4 c includes:

Now With **Pascal** Support!

★ **A Source Code Analyzer:**
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

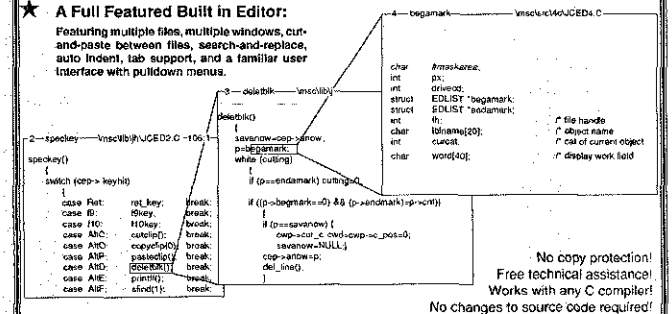
★ **A Point-and-Shoot Zoom Key:**
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ **A Full Featured Built in Editor:**
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pulldown menus.

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

- Philippe Kahn, "The Connection", Spring/Summer 1988

4 c does this and more!



No copy protection!
Free technical assistance!
Works with any C compiler!
No changes to source code required!
Required IBM PC compatible with 512K RAM.

Call or Write:

Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, Il. 60130
312-366-7595

30 Day Money Back Guarantee
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Number **276** on the Reader Service Card

VOICE MASTER KEY®
VOICE RECOGNITION SYSTEM
FOR PC/COMPATIBLES & TANDY 1000 SERIES
A FULL FEATURED VOICE I/O SYSTEM

GIVE A NEW DIMENSION TO PERSONAL COMPUTING... The amazing **Voice Master Key System** adds recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, desktop publishing, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. Voice recognition tool-box utilities are included. **A genuine productivity enhancer!**

SPEECH RECORDING SOFTWARE...Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data compression utilities. Create software sound files you can add to macros for voice recognition verification response, **a complete, superior speech and sound development tool.**

SOFTWARE CONVERSION CODES. The **Voice Master Key System** operates a growing list of third party talking software titles using synthesized phonetics (text - to - speech) or digitized PCM, ADPCM, and CVSDM encoded sound files. **Voice Master Key system does it all!**

EVERYTHING INCLUDED. **Voice Master Key System** consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. **High quality throughout, easy and fun to use.**

ONLY $149.95 COMPLETE

ONLY $89.95 FOR TANDY 1000 SL/TL MODELS SOFTWARE ONLY—REQUIRES TANDY BRAND ELECTRET MICROPHONE.

ORDER HOTLINE: (503) 342-1271 Monday-Friday, 8AM to 5 PM Pacific Time

Visa/MasterCard; company checks, money orders, CODs (with prior approval) accepted. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3½" or 5¼") when ordering. Add $5 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes. 30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED. ONE YEAR WARRANTY ON HARDWARE.

CALL OR WRITE FOR **FREE** PRODUCT CATALOG

COVOX INC.
675 Conger Street, Eugene OR 97402
TEL: 503-342-1271 • FAX 503-342-1283

Number **277** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PC Communications and Protocols

by
M. Steven Baker

A decade ago, data communications on microcomputers could be easily described—asynchronous serial I/O. Of course, you had your choice of either modems or direct lines to peripherals or other computers. Life was simple. But today, data communications spans the gamut from low-speed, asynchronous, serial I/O to fast, fiber-optic network connections. I'll limit my remarks here to PC serial communications and save network communications and protocols for a future article.

Hardware KIX with Serial Chips

I first cut my teeth on microprocessors writing serial I/O drivers in that far-distant land of CP/M-80. You had to start at the hardware level and build up. In those days, each microcomputer manufacturer made its own choice of the peripheral chips used to support communications with the outside world. The parallel interface was available but was not used as often. A letter-quality or dot-matrix printer was just as likely to sport a serial interface as the now-standard Centronics (IBM) printer interface. For many reasons, it was preferable to locate printers away from the computer. At home, my printers are in the next room, where any noise goes unnoticed. (Today, of course, laser printers are peaceful beasts.)

The commonly used UART (Universal Asynchronous Receiver/ Transmitter) chips included the Motorola 6850, the Signetics 2651, the Zilog Z80 SIO, and the Intel 8251. The last three chips supported both synchronous and asynchronous I/O. In fact, the 8251 was used on the IBM PC synchronous adapter, and a successor to the Z80 SIO has been used for the high-speed serial interface to Appletalk on the Mac. Even though only a handful of serial I/O chips were used, each manufacturer would locate these chips at whatever CPU port or, in some cases, at any memory-mapped device addresses it chose. A common task of the time was porting the latest version of CP/M communications software to the latest new 8085 or Z80 microcomputer. The number of machine-specific overlays for the de facto CP/M-80 modem programs number well over a hundred.

The IBM PC changed all that. IBM chose a UART chip from National Semiconductor (and Western Digital), the INS8250, for the two possible serial ports on the PC. Although the IBM ROM BIOS supported serial communications poorly, the hardware embodied in the 8250 chip became the standard. Unlike most of the 8080/Z80 world, the IBM PC ran its peripherals with interrupts. If you didn't want to lose characters, the IBM PC communication programs had to at least use an interrupt handler for receiving data. This became absolutely mandatory at the higher baud rates.

The 8250 chip supports only asynchronous serial I/O. Eight I/O ports are used for accessing control and data registers. The chip supports four different interrupts, in the following order of priority:

- Receiver line status (errors)
- Received data ready (got a CHAR)
- Transmit holding register empty (okay to send CHAR)
- Modem status (change in DTR, DCD, or Ring Indicator)

The 8250 chip family has gone through a number of revisions. National Semi has produced seven versions:

- INS8250
- INS8250-B
- INS8250A
- INS82C50A
- NS16540
- NS16C540
- NS16550A

First was the original INS8250 and the slower-speed (CPU bus timing) 8250-B. The software interface to the 8250 is well described in the IBM PC and AT technical reference manuals and a number of trade books. What is not documented are a number of bugs in these chips, particularly related to the transmit empty registers and transmit interrupts (see references). Fortunately, most programs (even modem programs) used polling for sending characters, not interrupts.

A few years back, I replaced a number of PC-clone serial boards for folks trying to run Windows 1.0 applications. Using serial printers, the Windows print spooler would enable transmit interrupts, and printing would die. A second major characteristic of this chip family was that it could not tolerate back-to-back I/O operations. Liberally sprinkled through the AT technical reference INT 15H serial I/O code are instructions like

```
jmp     $+2
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The purpose of this jump routine is to flush the 80x86 prefetch queue and generate enough of a delay to access the 8250 chip again.

The read-and-write cycle delay required of these earlier chips is 1725–1785 nanoseconds, more than ten times slower than the newest chip. The 8250A, an improved version with fewer bugs, required one-third the delay of the 8250. The later 16450 chips are used in 286 and 386 machines because they can be accessed faster than the original chips. The newest chip is the 16550A, which has separate transmit and receive FIFO buffers, a DMA interface, and faster CPU timings and also supports higher maximum baud rates. The most useful feature is the 16-character receive FIFO buffer that can be enabled. This nicely prevents the loss of characters at rates of 9600 baud and higher when running under multitasking software such as DESQview or OS/2 (on the 286). Unfortunately, the current crop of PC communications software doesn't yet support these advanced features. Also, most AT serial boards have the 16450 chip (at least the first one) soldered in, making a simple chip replacement very difficult.

These chips can easily be differentiated with some software tests. The 8250A and 165*X*0 chips possess an additional scratch register (register 7 at 3FFH for COM1 port). Write a bit pattern to this port and then read it back to distinguish these chips. This scratch register may be used for anything. In the 16550A, the read-only interrupt identification register (IIR) also becomes the write-only register for FIFO control. Setting bit 0 to 1 will enable both receive and transmit FIFOs. The 16550A uses the top two bits of the interrupt enable register (IER) to identify that the FIFOs are enabled.

Using a 1.8432-MHz crystal, these UART chips can be run at up to 112,000 baud (divisor of 1), although the PC BIOS supports up to only 9600 baud (a divisor of 12). Intermediate speeds of 19,200, 38,400, and 56,000 baud are potentially available. Some laser printers, terminals, and high-speed modems now support up to 38,400 baud.

PC File-Transfer Protocols

In communication between devices, protocols are usually necessary to define the framing for the information and guarantee error-free data delivery. Of course, when communicating with your printer, you assume that no errors occur; however, with ever higher speeds, that is likely to change. Already, most PC PostScript printers support the Appletalk protocol. High-end printers (IMAGEN lasers, for example) often sit on a network and have built-in Ethernet adapters.

Aside from printer protocols, the most familiar protocols used on the PC are for file transfers. These range from public domain protocols to those used in proprietary products such as Blast and Relay. Over time, public protocols have found the most widespread use. I will briefly comment on four of the most commonly used protocols:

- XMODEM and its derivatives
- KERMIT
- UUCP Packet Protocol "g"
- ZMODEM

I have included the actual protocol manuals on the code disk for detailed documentation.

A number of concerns are involved with the choice and the development of a protocol, including simplicity and ease of implementation; robustness and error recovery; speed and efficiency; and hardware and software limitations. Another important consideration in reviewing these protocols is to evaluate the *protocol* separately from its *implementation* in a specific application program.

XMODEM, the Misnomer

What is now called XMODEM is based on a CP/M-80 modem program written by Ward Christenson in 1977. Christenson has immense stature in my personal "PC Hall of Fame." In the early days of CPM and microcomputers, he was a prolific and talented programmer and writer. Unbeknownst to most IBM PC folks, Christenson authored not only what has become known as the XMODEM protocol but also the first versions of many of the best CPM utilities. These include RESOURCE (the model for most interactive disassemblers), DUU (a disk utility program still without equal in the PC world), CAT (a disk cataloging system), and scores of other programs. I learned how to program in 8080 assembly language from his fine columns in *LifeLines* magazine (published by Lifeboat Associates).

MODEM2—or MODEM, as it was called—was described much later by Christenson as a "quick hack." It was written in 8080 assembly language, as were most of the CP/M-80 programs of the day. At the time, the protocol was not refined or described except by the documentation in the source file. The protocol was designed for microcomputers that could pass 8-bit data (binary files) across a serial line or modem. The basic protocol was receiver driven and very simple. A basic single-character handshaking phase is followed by the file broken into data packets containing a special start character (SOH–01H), the block number (starting with 1), 255 minus the block number, 128 bytes of data, and a byte checksum. The receiver starts the handshaking by sending a single character—the NAK (15H) for the original program. A single EOT (04H) character is sent to mark the end of a file transfer.

Slightly revised and improved CP/M versions continued under various MODEM*X* names, but the protocol remained stable. By 1980, what we now call MODEM7 batch (multifile) mode was added, allowing sending a whole disk of files at once. The way names are passed is now considered "brain-damaged" (one character at a time waiting for an echo back from the receiver), but it is still functional. By 1981, the two-byte CRC-16 checksum had been added as an option to the original single-byte checksum. The receiver requests a CRC transfer with a "C" instead of an NAK.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The name "XMODEM" is a misnomer of sorts. XMODEM was the name for the remote modem program (based on the protocol) that would be invoked on a CP/M BBS system to send or receive files. The protocol is also known as the Christenson protocol or as MODEM7. By 1981, the complete protocol as we know it today was reasonably well defined (including CRC) and in widespread use. A VAX version of XMODEM written in FORTRAN was available by 1982. This was a time when the standard was 300-baud modems from U.S. Robotics and D.C. Hayes. The fastest affordable modems were PMMI S-100 boards, with speeds of up to 710 baud.

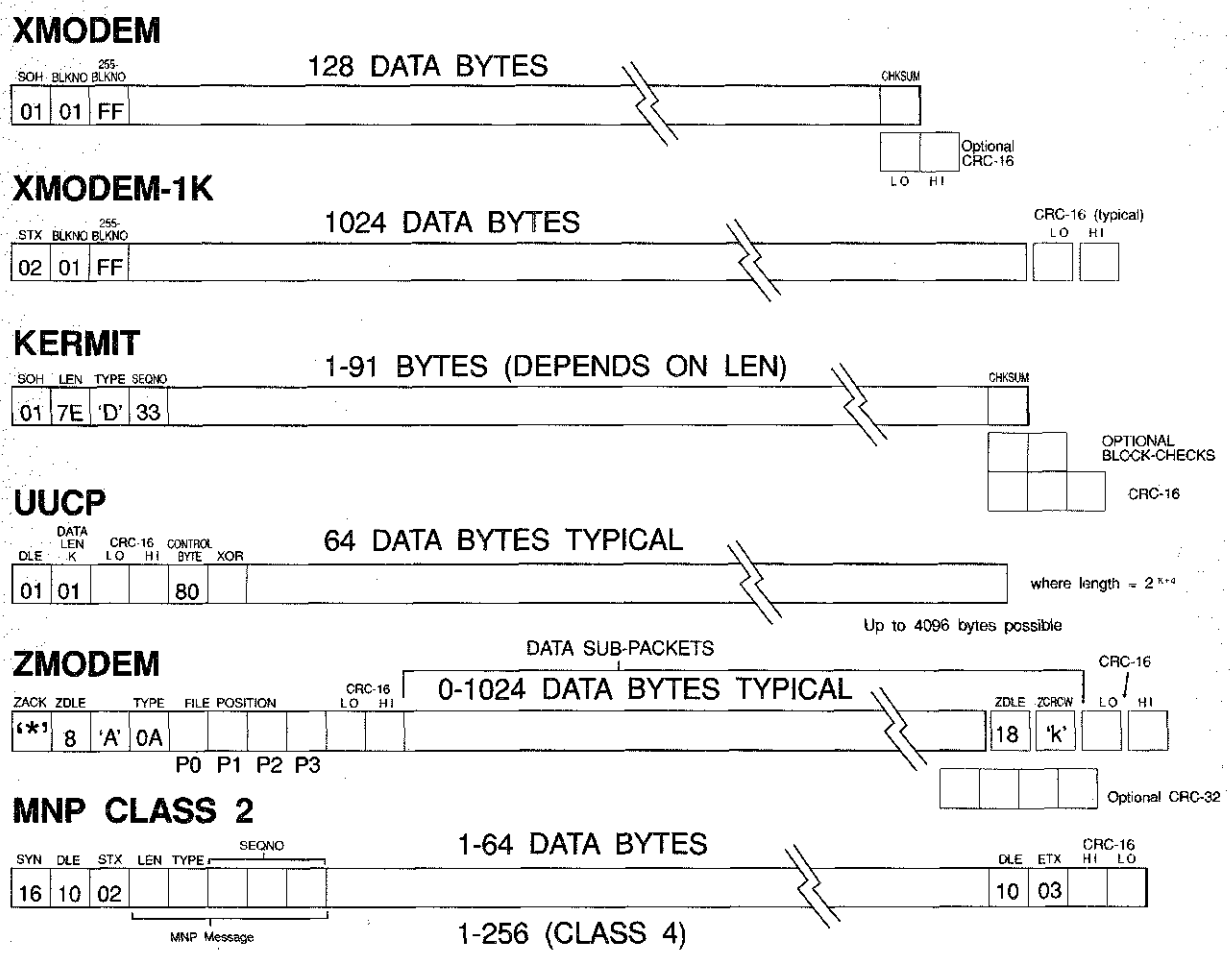
One nit I will pick is the substandard quality of early communications software on the IBM PC. The original IBM PC, with single-sided floppy drives and 16K of memory, was introduced in 1981. DOS 1.1 was introduced in 1982; the PC-XT, in 1983. While the CP/M-80 world had two years earlier switched over to XMODEM-CRC, Hayes's SMARTCOM II (and other PC programs and BBSs) supported only the ancient checksum XMODEM protocol well into 1986. When you read about the poor error checking of the XMODEM protocol, you should understand this to mean the poor *implementations* of XMODEM in PC software. I ported over the CP/M-80 MODEM901 program to DOS in 1983 for a number of my clients frustrated with the likes of SMARTCOM.

XMODEM operates as a start–stop, or half-duplex, protocol. The next block is not sent until the previous block has been acknowledged. XMODEM uses a packet with a checksum only for data blocks. ACKs and NAKs, being a single character, are more susceptible to line damage. Because of the small packet size and the start–stop nature of the protocol, the efficiency of XMODEM transfers will drop at higher baud rates and over packet-switched networks (Telenet) that delay characters. Remember that XMODEM was designed when 300 baud was the standard and files to transfer were small (CP/M programs written in assembly language to run in 64K of memory). CPM files were defined as so many 128-byte blocks. Based on CP/M, XMODEM file transfers are rounded up modulo 128. Since CP/M-80 2.0 didn't support file date and time stamps, neither does XMODEM. On the other hand, the XMODEM protocol is extremely simple and easy to implement. Even under CP/M, only a small portion of the latest MODEM*XXX*-version source code deals with the basic file-transfer function.

XMODEM Derivatives

The first problem that needed fixing in XMODEM was the small, 128-byte data-block-size limit at faster baud rates. This is where the protocol confusion starts. This particular change went in several directions. Two different CP/M camps added the 1024-byte-block data option to the basic XMODEM protocol. The actual data-packet protocol is the same; the STX (02H) character begins the header of a 1024-byte data packet instead of the SOH character. However, the camps differed on how the larger packet size was negotiated.

Sample Data Packets (in bytes; values are normally HEX)



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One group insisted on a two-character sequence ("CK") from the receiver, while the other, larger camp requested that the sender force the larger block size via a command-line switch. Both of these protocol derivatives have become known as XMODEM-1K.

Another complication was caused by a solution to the second problem. XMODEM was developed for a CP/M system without date/time stamps or exact file sizes. Chuck Forsberg, the author of an early CP/M and UNIX modem program in C (YAM for Yet Another Modem program), had enhanced XMODEM by adding a data block 0, which contained the filename, date/time stamp, exact file size, and file attributes. This protocol also supported the 1K-packet option and became known as YMODEM batch, a successor to XMODEM. Again however, the implementation in many PC communications programs was often substandard or misnamed. YMODEM could mean just the XMODEM-1K protocol or the real YMODEM batch protocol in BBS and communications software. The YMODEM batch protocol as defined by Forsberg also included a more sensible batch file transfer feature.

By the addition of other characters (sequence numbers) to the ACKs and NAKs, possible damage to the single-character negotiation and acknowledgement characters has been addressed by some other derivatives—Sealink, Telink, Megalink, and WXMODEM. The final problem with the start–stop nature of the protocol was also attacked in several directions. A sliding-window version of XMODEM, called WXMODEM, has found limited use. Rather than wait for an acknowledgement for each packet, a window slides along that allows up to four packets to be sent without waiting for an acknowledgement. Each packet acknowledgement adjusts this sliding window.

In another direction, a *g* option was added to the YMODEM protocol to eliminate the waiting (the receiver sends a "G" to initiate it). This assumes the file is sent via some error-correcting modem or media. If an error occurs, the receiver sends an NAK and the entire transfer aborts. Other XMODEM derivatives—BIMODEM (a full-duplex protocol), JMODEM, and others—have been proposed and developed, yet none has enjoyed the widespread support of the original XMODEM protocol.

KERMIT—Celtic for "Free"

Developed by Frank da Cruz and Bill Catchings, of Columbia University, in 1981, the KERMIT protocol (named after the Muppet character) was designed specifically to facilitate communication between PCs and mainframes. The protocol was constructed to deal with the restrictions and peculiarities of various computer environments—file structure, character set, buffering, parity, and so on. The protocol was based on a layered design.

The basic protocol is character oriented and uses only printable characters, with the exception of one MARK (start-of-frame) character. This MARK character defaults to SOH (Ctrl-A) but is dynamically configurable by most implementations of the protocol. KERMIT uses packets for all information transfers. KERMIT programs on each machine send packets back and forth. The KERMIT protocol needs only minimal requirements to function. KERMIT does not require any particular baud rate, hardware or software flow control, full-duplex systems, or 8-bit serial data capability.

The protocol supports transfers of two types of files: printable text and binary. Communicating with host computers

Software Tools
For Programmers & Non-Programmers

Opt-Tech Sort™

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can read the data. Example: 1,000 records of 80 bytes can be read sorted and a new file written in less than 9 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or Called as a subroutine to over 30 programming languages.

Numerous features are included. A partial list of features includes: The ability to process files of any size. Many filetypes are supported including Sequential, Random, delimited, Btrieve, dBASEII & III and many others. Up to 30 key fields can be specified over 16 different types of data are supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes. **Newest Features: Faster than ever! LIM EMS support. User exits to pass or receive records. Less disk space required and much more!** MS-DOS $149.

On-Line Help™

On-Line Help allows you to easily add Help Windows to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable help systems.

The resident version allows you to add help to any system. Your Help system is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire. Upon clearing the help windows you are right back into what ever program you left.

The callable version allows you to easily display help windows from your programs. A simple Call to the help system makes the window appear. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location. MS-DOS $149. Demo $10.

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. Its a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing
P.O. Box 678 / Zephyr Cove, NV 89448
(702) 588-3737

Number 16 on the Reader service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

supporting only 7-bit serial data, binary files may be sent using an optional prefix encoding. The basic protocol supports nine packet types:

D Data
Y Acknowledge (ACK)
N Negative acknowledge (NAK)
S Send initiate (exchange parameters)
B Break Transmission (EOT)
F File header
Z End of file (EOF)
E Error
T Reserved for internal use

Any nonprintable characters required for sequence numbering, parameters, and checksum information are converted to printable characters, usually by adding 32H, hence the 96-character limit (LEN byte = 94) on total packet length. Control characters are escaped by a quote byte, usually "#", and converted to printable characters. KERMIT uses the SEND-INIT and the responding ACK packet to pass parameters between the two KERMIT programs. These options establish the maximum packet length to be received, the number of seconds to timeout waiting, the number of padding CHARs to precede each packet, the control character needed for padding, the CHAR needed to terminate an incoming packet, and the printable ASCII character used to quote control characters (normally "#"). Other optional features include prefix quoting for bytes with the eighth bit set, two-byte checksum or three-byte CRC-16 check types, run length encoding of repeated characters, and a special-capabilities mask for such things as passing file attribute packets.

KERMIT has one great virtue: its ability to reliably transfer files between PCs and mainframes. Until recently, KERMIT has been the only public domain protocol that addressed this problem adequately. Implementations of the KERMIT protocol exist for the largest variety of computers imaginable implemented in languages as diverse as APL (Burroughs) and PL/I (Prime). The better mainframe KERMIT implementations support a "server mode" that allows all file-transfer commands to be given from the remote (typically PC) computer. As a service, Columbia distributes magnetic tapes and floppy disks of various KERMIT programs for modest fees. These versions are also available for downloading over BITNET or ARPANET.

Because of the small data-block size (91 bytes max), efficiency at high speeds and over packet-switching networks (such as Telenet) is poor. On 8-bit data streams, XMODEM will be faster, because of the larger packet size and single-character acknowledgement. In 1986, extensions were added to the basic protocol to address these efficiency problems. These extensions support long packets (up to 9024 bytes), extra-long packets (up to 857,374 bytes), sliding windows, detailed information on file types, passwords, data encoding, and other features. These new extensions (particularly sliding windows) increased the complexity of the basic protocol. Although the window KERMIT protocol has been around a few years, many of the older mainframe and microcomputer implementations don't support it yet.

UUCP Packet Protocol "g"

The UUCP program was developed in 1978 to run on AT&T Bell UNIX systems to facilitate software distribution and exchange (file transfers and mail). The term *UUCP* is often used to describe the family of programs written by Nowitz and Lesk that work together to provide these basic networking features under UNIX. My comments will instead focus on the "g" packet protocol developed by Greg Chesson (from whence I assume comes the *g*), formerly with Bell Labs.

The UUCP PK, or packet driver, ("g" protocol) was designed for 8-bit byte transfers. After an initial login, password, and negotiating work phase, the UUCP file-transfer protocol sends packets back and forth between the two computers. The packets are framed with a six-byte envelope—a start byte (DLE - 18H), a four-byte message, and a byte that is the exclusive OR of the four message bytes. The message consists of a type byte, a two-byte checksum of any following data segment, and a control byte. Packets can consist of three distinct types—control packets (seven possible control messages), data packets, or short data packets. Data packets have a data segment following the basic packet. The allowed data block sizes range from 32 to 4096 bytes and are constrained by the values from an initiation parameter exchange. Short data packets contain count bytes and are designed to handle smaller data segments. The protocol depends on the concept of a transmission window (sliding window) to define the upper and lower bounds for valid data-packet sequence numbers. This sliding window can range up to seven packets.

The control-packet encoding is very efficient: One control byte contains a complete message and is partitioned into three fields:

bit	7	6	5	4	3	2	1	0
	t	*t*	*x*	*x*	*x*	*y*	*y*	*y*

The *tt* bits indicate the packet type and determine how the *xxx* and *yyy* fields are to be interpreted. The packet types are as follows:

tt	*interpretation*
00	control packet
01	alternate channel (obsolete)
10	data packet
11	short data packet

For control packets, the *xxx* field is interpreted as a control message as follows:

xxx	*name*	*yyy*
1	CLOSE	no meaning
2	RJ (reject)	last correctly received sequence number
3	SRJ (selective reject)	sequence number to retransmit (obsolete)
4	RR (receiver ready)	last correctly received sequence number
5	INITC	window size (1-7)
6	INITB	data segment size 32*2^yyy
7	INITA	window size

Sequence numbers are passed modulo 8. Data packets use *xxx* for the current sequence number and *yyy* for the newest sequence number accepted by the receiver. UNIX implementations of the protocol generally limit the data-segment size to 64 bytes (in order not to overrun statically allocated buffers). My limited experience with UUCP indicates that a sliding window of three packets is common. This may have been adequate for direct-connect lines and slower modems, but

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the newer, high-speed modems with turnaround delays can benefit from a larger sliding window. The PK protocol is more complex than XMODEM but does not appear difficult to implement. The protocol efficiency using the sliding window should be good even over packet-switching networks.

ZMODEM

In 1986, Telenet (the PC Pursuit folks) funded Chuck Forsberg, the author of the YMODEM batch protocol extensions to XMODEM, to develop an improved public domain file-transfer protocol. This protocol was to be designed to alleviate the efficiency problems network customers were having with XMODEM and KERMIT file transfers. The initial concept was to merely make modifications to XMODEM for high performance. Fortunately, Forsberg ultimately decided to design and implement a revised protocol based more on the spirit of XMODEM, using the best ideas from all of the existing protocols. He also added some truly useful new features not found in other available protocols. This new protocol was called ZMODEM, as a successor to X/YMODEM, even though ZMODEM looks nothing at all like XMODEM. The ZMODEM protocol documentation and implementations in C for UNIX machines (*sz* and *rz* programs) were placed in the public domain (and are on this issue's code disk).

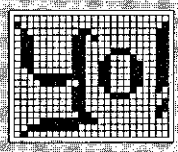
ZMODEM is a packet protocol; that is, programs send ZMODEM packets or frames back and forth. One exception, the CANCEL sequence, 5 CANs (18H), could actually be considered a special packet. From KERMIT and UUCP came the concept of an initial dialog to exchange system parameters. The exchange of parameters allowed supporting both binary (for efficiency) and hex (application-friendly) frames. From BISYNC and HDLC came the concept of variable-length data packets (up to 1024 data bytes) rather than the fixed-block XMODEM packets. Data packets begin and end with an escape sequence. The sliding-window protocols (window KERMIT and UUCP) were taken one step further by making the entire file one big sliding window for streaming. Error protection was enhanced with an optional CRC-32 four-byte checksum.

The ZMODEM protocol requires an eight-bit transfer medium but escapes network control characters to allow operation over packet-switched networks. ZMODEM places no restrictions on binary file transfers. The ZMODEM protocol uses a binary or hex header followed by various other subpackets. Twenty different packet types are defined similar to KERMIT. The ZPAD ('*') character is used to mark the start of a packet. The most outstanding ZMODEM feature is its check–restart capabilities. The sliding window actually consists of the entire file. If a transfer is aborted for any reason, the transfer can be restarted from that part of the file that was missed—you don't have to start over from the beginning. The ZMODEM protocol is very fast and works well over packet-switched networks. Error checking is excellent.

Because of its efficiency, ZMODEM is fast becoming one of the most commonly used file-transfer protocols on PC BBS systems. Although ZMODEM has been implemented in only a few programs on the PC, it is intensively used by other communications programs (PROCOMM, for example) by shelling out to DOS and running DSZ, a file-transfer protocol engine (shareware). Also written by Forsberg, DSZ supports ZMODEM, YMODEM, and XMODEM. The ZMODEM protocol is somewhat complex;

Quality Programming Tools From
Ithaca Street Software, Inc.

FONT-TOOLS



If you develop graphics applications that include the use of text then you should definitely own Font-Tools. A complete solution to your graphics text needs, Font-Tools includes a full-screen font editor, utilities for modifying, viewing, downloading, and format-converting fonts, over 60 high quality sample font files, and a high performance library with over 40 C callable functions for drawing text on the screen or into memory buffers. This product's features are too numerous to mention, so call us or write for more details. We think you'll be pleasantly suprised.

Single User $99.95 - Developer's Ed. $149.95

BABY DRIVER

Baby Driver's superb mix of features, versatility, and performance has it slated to become an industry standard for graphics printer support. It currently supports more than 150 dot matrix, inkjet, and laser printers, and additional printers are being added on a continuing basis. Its linkable object libraries allow for seamless integration with your graphics application while adding a mere 12K to its size. We've just added function support for PCL and support for true color printing is scheduled for release this summer. All things considered, a must for the serious developer.

Single User $99.95 - Developer's Ed. $199.95

ICON-TOOLS/Plus

If you think of icons as being only a specialized kind of font then you've never seen Icon-Tools. Its icon editor will remind you of our Font-Tools editor, but the similarities end there. Icons can be used to build graphics menus, and the Icon-Tools function library supports this use. Since it is customary to use a mouse in this setting, we've added mouse support. We've also added functions to produce sound and animation effects. The sample icon files provide more than 500 icons for your use. Try it and see if you agree with us that this is a nifty product.

Single User $99.95 - Developer's Ed. $125.00

1145 Ithaca Drive, Boulder, CO 80303
phone (303) 494-8865 fax (303) 494-7802

Number 226 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

MNP Protocol Classes

Class 1: The basic error-correcting protocol for asynchronous, byte-oriented data exchange in half-duplex (one-way) mode. The protocol efficiency is about 70%.

Class 2: Asynchronous, byte-oriented data, full-duplex data exchange with error-correcting protocol. The protocol efficiency is about 80%. Maximum protocol data packet size is 64 bytes.

Class 3: Strips start and stop bits and uses synchronous bit-oriented, full-duplex exchange with the error-correcting protocol. The computer still sends asynchronous data to the modem. The protocol efficiency is about 108%.

Class 4: The maximum protocol data packet size is increased to 256 bytes. The MNP protocol monitors the reliability of transmission. On relatively error-free channels, MNP assembles larger data packets to increase throughput. Some redundant MNP information is reduced from packets. The protocol efficiency increases to about 120%.

Class 5: Basic data compression is incorporated to MNP Class 2 or higher using adaptive tokening and run-length encoding. Variable-length bit strings are dynamically substituted for bytes by both sender and receiver. The bit string assigned to any byte will change over time on the basis of the most recent probability of bytes. A typical compression factor is about 1.5 to 1.6, resulting in a protocol efficiency of 180 to 200%.

Class 6: Features are added for negotiating speeds upward for half-duplex (V.29) modems and dynamically allocate the data channel to simulate full-duplex service.

Class 7: Enhanced data compression is added using true huffman encoding and run-length encoding. The compression factor is typically 2.4 for uncompressed files, and protocol efficiencies in some cases can approach 300%.

Class 8: Enhanced data compression combined with V.29 half-duplex technology.

Class 9: Enhanced data compression applied to V.32 full-duplex modem technology. Protocol efficiency can reach 300% in some cases, compared with normal V.32.

a programmer would need both the protocol manual and the sample *sz* and *rz* program sources for implementation. The current sample C sources include files to implement the ZMODEM protocol on the VAX and UNIX.

Modems at Blazing Speeds

Several developments continue to take place in the modem market, including ever higher speeds and error correction. With the development of digital-signal-processing (DSP) chips and lower-cost, more-powerful VLSI and CPU engines, modem speeds are extending the limits of the analog bandwidth over phone lines. On top of these *data pumps,* the modem CPUs have added their own error-correcting protocols along with data compression. Basically, four types of high-speed data pumps are now in use by various modem manufacturers:

- CCCIT V.29 half-duplex 9600 baud (Microcom, Hayes, etc.)
- Asymmetric modems with 14,000-baud and 450-baud dual channels (U.S. Robotics Courier HST)
- Multicarrier-frequency modems (Telebit Trailblazer)
- CCCIT V.32 full-duplex 9600-baud modems

CCCIT is the international organization that adopts voluntary standards for modems and other communications equipment. The V.29 standard is older, 9600-baud technology that has become standard in fax machines. By its nature, it is half-duplex; so, the line direction must be turned around when data must be sent in both directions, requiring a minimum of 50 milliseconds. These modems are often called "pseudo full duplex"—they simulate full duplex by turning the line around. For file transfers with small data blocks, this turnaround time can adversely affect efficiency; with larger blocks, this problem is alleviated. Also, for single-character input and echo, overall response with the turnaround times may actually be worse than in regular 2400-baud, full-duplex modems. But for full-screen updates and large block-file transfers, these modems will deliver solid performance. Under adverse line conditions, most V.29 modems will fall back to 7200, 4800, and 2400 baud.

The asymmetric modems (Courier HSTs) are full duplex, but the two channels operate at different speeds: 14,400 and 450. This design allows for greater efficiency for file transfers where most traffic is in one direction. But for single-character input and echo, the line-turnaround time to reverse the direction twice is a minimum of 250 milliseconds. Again, these modems fall back in speed to 9600, 7200, 4800, and, ultimately, 2400 baud on poor lines. A CCCIT (V.34) study group has been meeting in regard to this design. Courier modems have been very popular with BBS SYSOPs because of special discounts provided by U.S. Robotics.

The multicarrier design (implemented by Telebit) breaks the available bandwidth into more than 500 little pieces for encoding, which can deliver up to 14,000 baud. This design is also a half-duplex scheme. The real strength of multicarrier is that the frequency chunks are allocated dynamically. If certain bandwidths on the line get noisy, the speed can be dropped in units of 100 baud. On your important call to India, where all of the other modulation techniques may fail, the multicarrier may get through, albeit at some low baud rate. The line-turnaround time is a minimum of 11 milliseconds (Telebit), and a single-character input and echo can take 120 milliseconds. Trailblazers have been very popular on UNIX systems and Usenet.

V.32 is the most recent CCCIT modulation standard. This technique has been around for years but available only in very expensive equipment. Now, all of the high-speed modem manufacturers are offering V.32 modems, many as dual standards supporting both V.32 and their other technology. Rockwell and others offer a chip set fully implementing the V.32 engine. V.32 is a full-duplex, 9600 modulation scheme using both quadrature amplitude modulation (both frequency and amplitude vary), trellis encoding, and echo cancelation. Being full duplex, it requires no line-direction turnarounds. Since it is standardized, one high-speed modem can now speak to another manufacturer's high-speed modem. On noisy lines, V.32 drops to 4800 baud—and in most manufacturers' products, to 2400 baud. The implication is that over clean lines you get excellent performance. But the fallback rate is steep over less-clean lines. Is this the new speed limit? Well, British Telecom already has 12,000-baud, full-duplex modems using these same techniques, with 14,000 baud promised.

Hardware Protocols

At 9600 baud and above, all of these modems depend on hardware (RTS/CTS) or software (XON/XOFF) flow control from the computer to the modem. This prevents the

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

modem's data buffers from being overrun. Software flow control often doesn't work reliably at the higher speeds, because of line delays. On top of these high-speed data pumps, manufacturers have added error correction for reliability and data compression to deliver even greater throughput.

Proprietary protocols have been developed, including the Packetized Ensemble Protocol (PEP) from Telebit. PEP uses a sliding window and a 16-bit CRC. But the most popular hardware protocols have been developed by Greg Pearson of Microcom: the Microcom Networking Protocols (MNP). MNP was developed over time, starting with a Class 1 and now going through Class 9 (see sidebar). Classes 1–4 are now in the public domain, and documentation is available from Microcom ($100). Microcom licenses MNP classes 5–7 for a one-time cost of $2500. The higher classes are currently available only on their own modems. The MNP protocol was designed from the start to allow for enhancement. Unknown parameters and messages are disregarded so that the older classes can still communicate with the newest MNP modems. The MNP protocol classes are negotiated individually so that some of these feature can also be implemented in software.

MNP classes 1–4 provide point-to-point error control. The error-correcting protocol in Class 2 actually adds overhead. Class 3 is synchronous between the modems, stripping start and stop bits for a twenty-percent improvement. Class 4 increases the maximum data size to 260 bytes. Class 5 added data compression, using an adaptive tokenizing scheme similar to huffman encoding. The most frequent characters are mapped to varying-length bit strings, some shorter than eight bits and some longer. (This mapping continues at both ends dynamically.) The current de facto standard for new 2400-baud modems is to incorporate MNP classes 1–5, delivering up to 4800 baud throughput on certain file types. MNP is also included with a number of V.29, and the newer V.32, 9600-baud modems. With data compression on certain files, these modems can double this throughput. Class-5 compression is not advantageous for already compressed ARC or ZIP and will commonly reduce throughput. Some modems allow disabling Class 5 with a Hayes-style AT command just for this purpose.

CCCIT officially adopted a V.42 error-correcting protocol standard in November 1988. Two protocols must be supported for a V.32 modem to be conforming—MNP classes 2 and 4 and LAPM (an X.25 derivative). CCCIT is also completing work on V.42bis—a standard for data compression not limited to modems. V.42bis has adopted a Lempel–Zev compression scheme put forth by a member of the British Telecom delegation.

Protocol Implementations

The high-speed modems must communicate with the PC at 19,200 or 38,400 baud—a pretty fast clip. The higher modem speeds are all for naught if your software communication protocol stops and waits. YMODEM "g" streaming or one of the window protocols becomes essential. On modems that may need to reverse the line direction, small data-packet sizes will also hurt performance. Telebit has a novel way of handling these software protocol problems. The modem itself supports several software protocols—XMODEM, YMODEM, KERMIT, and UUCP "g". Using a Hayes AT-style command notifies the modem of the software protocol to be used. The modem responds to the sending PC with the appropriate protocol, strips off the software protocol information, and repackets the data with its own sliding window and error-free protocol. The result is that the software protocol chosen no longer adversely affects performance.

There can also be differences in the implementation of a software protocol from one program to another. For example, I found that KERMIT transfers to my local minicomputer (PRIME) were ten to twenty percent faster using MEX (a

MISOSYS

Programmer's Journal downloads are now available from MISOSYS' forum on Compuserve: GO PCS49

HartFORTH-86™

HartFORTH is a full 79-STANDARD FORTH. It is a Direct Threaded Code implementation which provides an execution speed between 10% and 40% faster than the classical Indirect Threaded Code implementation. HartFORTH is designed to run under DOS so the Virtual Memory that it accesses for storage and retrieval purposes is a file created and controlled by the operating system. Enhancements to the 79-STANDARD have been built into the HartFORTH kernel in the form of functions to call the DOS file handling routines so that other files may be created and accessed if required. A library of standard screens is supplied with HartFORTH to provide often used extensions to the language, such as double length and floating point math, editing of source screens, string manipulation, arrays, etc.

Here's the FINE PRINT

Any MISOSYS software product purchased directly from MISOSYS may be returned ***within 30 days of the invoice date*** *for a full refund of the product purchase cost. Returns must be in a "like-new" saleable condition and be accompanied by a copy of the original invoice.*

HartFORTH-86
$59.95 + $5S&H (US)

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS or 703-450-4181

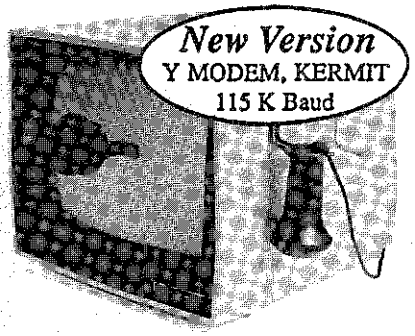
- HartFORTH programs can invoke other programs via **EXEC** and **EXEC.PROG**.
- Functions create new files from within HartFORTH, and allow the current Virtual Memory file to be changed for another and manipulated at the individual block level.
- Provides the recommended 79-STANDARD DOUBLE NUMBER STANDARD EXTENSION word set that implements 32-bit operations.
- **CASE:** and **SWITCH:** functions allow multi-way branching decisions to be taken with execution continuing in-line once the word branched to completes.
- String manipulators include: `"VARIABLE`, `"CONSTANT`, `"!`, `"LEFT`, `"RIGHT`, `"MID`, `"+`, `"COMPARE`, and `">`.
- DOS software and hardware interrupt vector access support via: `GET.VECTOR`, `PUT.VECTOR`, `THIS.SEG`, `DI`, and `EI`.
- V24 program input, output, and interrupt input support.
- *Overlay management words:* `FORGET.OVLY`, `OVLYNAME`, `PUT.DATA`, `OVLY.ENTRY`, `SAVE.OVLY`, `CORRECT?`, `LOAD.OVLY`, `NEW.OVLY`, `RUN.OVLY`, and `LEAVE.OVLY`.
- Screens provide trigonometric functions: `SIN.`, `COS.`, `TAN.`, `SIN`, `COS`, and `TAN`.

Number 121 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

We Have Ways To Make Your Computer Talk

C Communications



C Communications Library Has On-Line Debugger

No Computer Is An Island

As we move closer and closer to the paperless society, more people are looking for ways to move information electronically. Essential Communications is the unsurpassed technique for accomplishing that goal in C.

Like all our products, Essential Communications is a comprehensive, completely debugged professional programming tool.

No Assembly Required

Now communicating with other PC's, mainframes, plotters, digitizers, modems or any device that utilizes the RS 232 port is as easy as turning a crank.

Communications programming requires controlling things down to the bit level, a tedious and dangerous job. Our functions allow you to add professional communications without previous knowledge of communications, without loss of data.

You can do all of this without resorting to assembly language. Something you probably don't want to do, even if you know assembler.

We Interrupt This Advertisement To Bring You A Bulletin

Included in the library is a functional bulletin board system (BBS). This system can teach you a lot about communications, and has formed the core of many BBS systems around the country.

We also include a terminal program. Both programs include the source code. The BBS alone is worth the price of the package.

Making The Connection Is Only Half The Problem

Writing communications has its own unique set of problems concerning debugging. Normal debugging techniques are not possible with communications programs.

Is it a software problem? Is it a hardware problem? Am I sending what I think I'm sending? Are the protocols correct? I know, it must be a modem malfunction!

The above products are trademarks of **South Mountain Software Inc.**

Reach Out And Debug Something

Our combo package includes BreakOut, an interactive on-line data monitor that helps isolate the problems mentioned above.

Giving someone a sophisticated set of communication functions like the Essential Communications Library without an adequate method of debugging just wouldn't be fair.

Power vs Glory

There has always been a trade off in this industry between ease of use and power. Our functions do not require a lot of setups, are well documented and most of all, thoroughly debugged. Essential Communications ease of use stems from our thoughtfulness and not from a lack of power. As with all our products we explain what we are doing every step of the way. Our support staff are all competent C programmers who are thoroughly prepared to assist you after the purchase.

Essential Communications Functions include, general functions, xmodem, timer, keyboard, video, imodem and interrupt services.
Source Code Included, 30-Day Money Back Guarantee.

Essential Communication ..... $249.
BreakOut ..... $125.
/resident_C/ ..... $199.

Multple Ports - Up to 8 simultaneously
XMODEM CRC/ Checksum/1K
Speeds up to 38.4K Baud
Recieve and transmit are interupt driven
Background communications with /*resident_C*/
Multiple XM0DEM sessions
Other Essential Products Include:
ScreenStar - Essential Graphics, Utilities

1-800-451-6174

SOUTH MOUNTAIN SOFTWARE™

Formerly Essential Software Inc.

South Mountain Software Inc.
South Orange Plaza
76 S. Orange Ave., Suite 3
South Orange, N.J., 07079
N.J. 201-762-6965 Fax 201-762-0118

Number **10** on the Reader Service Card

commercial PC program from NiteOwl Software) than with the Columbia PC implementation of KERMIT. Columbia KERMIT uses small, 2K data buffers; so it is constantly going to disk. Even the old CP/M-80 modem programs used 16K buffers, and MEX will use up to 64K for data buffers.

What programs do I use myself, you might wonder. I consider communications to be like program editors: Once I get used to something and it works, I am reluctant to change. I use an old version of MEX-PC for most PC-to-BBS communications, and nowadays I shell out to DSZ for ZMODEM file transfers. Columbia KERMIT is my mainstay for terminal emulation and file transfers to IBM, VAX, and other mainframes. Columbia KERMIT has an excellent VT100 terminal emulator along with a TEK4010 graphics emulator. The latest PC version is 2.32A. Basic KERMIT may be slow, but it's reliable and readily available. At 19,200 baud with a direct connect line, KERMIT transfers were a dismal 200 to 250 characters per second between PCs and a PRIME mini (old KERMIT implementation). My next project is to port ZMODEM and the window KERMIT (written in C) to our local PRIME mini to speed up these file transfers. But in the meantime, I'll be more than grateful to use what I have. PJ

Steve Baker is the editor of Programmer's Journal.

References

KERMIT: A File Transfer Protocol, by Frank da Cruz, Digital Press, Bedford, Mass. (1987).

KERMIT Distribution, Columbia University Center for Computing Activities, 612 West 115th Street, New York, N.Y. 10025.

Microcom, 500 River Ridge Dr., Norwood, MA 02062, 617-551-1000.

Microcommunications Elements Databook (soon to be out of print), *Data Communications/Local Area Networks/UARTs*, National Semiconductor, Santa Clara, Cal. 408-721-5000.

NiteOwl Software, Route 1, Box 7, Fort Atkinson, WI 53538, 414-563-4013 voice, 414-563-9361 BBS.

Telebit, 1345 Shorebird Way, Mountain View, CA 94043, 800-835-3248.

US Robotics, 8100 N. McCormick Blvd., Skokie, IL 60076, 800-342-5877.

The ZMODEM Interapplication File Transfer Protocol, Omen Technology (Chuck Forsberg), 17505-V N.W. Sauvie Island Rd., Portland, OR 97231, 503-621-3406 voice, 503-621-3746 BBS.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

What Is C++?

(A Videotape Presentation)

Considering C++ for your next project? Confused after trying to read a C++ book? Or just curious about all the hOOPla? This VHS-format videotape, featuring author/speaker William M. Miller, gives a comprehensive and understandable overview of the major features of C++ and Object-Oriented Programming.

- On-line access to lecturer (via BIX)
- Moneyback guarantee!

Send $19.95 (+$4.00 S&H) for 1-week rental or $59.95 for purchase (postpaid) to:

Software Development Technologies, Inc.
Dept. 402
P. O. Box 366
Sudbury, MA 01776

Or call (508) 443-5779, 8:00 a.m. to 6:00 p.m. Eastern time, for Visa/Mastercard orders.

Inquire about our full C++ training course, available for on-site or videotape delivery.

Number **275** on the Reader Service Card

Parallel Programming for "C"

INTERWORK™

A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for simulation, real-time applications, and experimentation with parallel programming.

FEATURES

- Supports a very large number of tasks (typically more than 100) limited only by available memory. Low overhead per task results in very fast context switching.
- Provides a full set of inter-task communication (ITC) facilities, including shared memory, locks, semaphores, blocking queues, and UNIX™-style signals. Also has building blocks for constructing your own ITC facilities.
- Handles interrupts (DOS version) and integrates them into task scheduling. Supply your own interrupt handlers or block tasks on interrupts.
- Lets you trace task switches and inter-task communication.
- Comes with complete documentation including a user's manual and reference manual of commands.

Interwork is available for the following systems:

Hardware	*Operating System*	*Price*
IBM PC, XT, AT	PC-DOS 2.0 or later	$129
IBM PC AT	XENIX™	$159
DEC VAX™, SUN	UNIX 4.2BSD	$249

PC-DOS version is compatible with DeSmet, Lattice, and Microsoft C compilers.

Please specify hardware and operating system when ordering. Shipping and handling included; COD orders add $2.50. Send check or money order to:

Block Island Technologies
Innovative Computer Software
13563 NW Cornell Road, Suite 230, Portland, Oregon 97229-5892
(503) 241-8971

Trademarks: Interwork, Block Island Technologies; UNIX, AT&T Bell Laboratories, Inc.; XENIX, Microsoft, Inc.; VAX, Digital Equipment Corporation

Number **244** on the Reader Service Card

C™ Programmer's Toolbox

PC/MS DOS 2.1 & later

CFLow™	Determine your C program's function hieararchy; external R/T functions; etc.
CPrint™	Beautify C source code to your selected format
PMon™	Monitor your program/OS execution; Determine where time is being spent...
	+ 12 more tools...

Volume I

Are you creating better, faster, higher quality and easier to read programs in a fraction of the time?

Join our many beginning and expert users.

Volume II

9 more tools that you requested including:

CLint™	Perform additional syntax checks of your C source code-find silly mistakes...
CritPath™	Determine your C program's critical path from PMon and CFlow report inputs...
CXRef™	Trace your symbol/variable usage; identify potential usage problems...

The Toolbox works with all compilers and libraries. Specify any number of files through wildcards and/or indirect files.

Buy both Volumes I & II for $140, saving $20, or only one volume for $79.95. CA residents please add applicable tax. Overseas, please add $10 shipping & handling. With our 30 day money back guarantee, you can't lose.

Call or write us today

MasterCard® VISA®

C™ Programmer's McTool Series

Apple® Macintosh™

Common Tool Features

- Powerful, built-in editor
- Automatic/manual search & replacement
- Selective C source code highlighting
- Works with all C compilers
- ANSI support

#1 **McCPrint™**	$59.95
#2 **McCLint™**	$99.95

MMC AD Systems™

Box 360845
Milpitas, California 95035
(408) 263-0781
"The C Tool Specialists"

Number **94** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Line of Code

by
John Otken

This is the first in a series of articles on assembly language programming for the PC.

Assembly language remains one of the most popular PC programming languages because there is no more-efficient method for expressing many algorithms. Unfortunately, the language is also saddled with the perception that coding becomes exponentially more difficult with program size. The truth is that large assembly language programs are no more difficult to write and maintain than those using high-level languages if you follow proper programming techniques.

This article provides techniques that can enable a programmer with a few years of experience to use assembly language effectively. It will cover information critical to larger projects, information that is difficult to find in available literature. I shall presume the reader has knowledge of the 80X86 instruction set, has some experience with an assembler, and is fairly proficient in a high-level language. My intention is to get you over the hump that prevents many people from using assembly language. Emphasis is on programming efficiency, resulting in more code in less time.

Requirements

You will need an assembler (either OPTASM, TASM, or a recent version of MASM), a debugger, and an MS-DOS "bible," such as Terry Dettmann's *DOS Programmer's Reference* (Que) or *Advanced MS-DOS*, by Ray Duncan (Microsoft Press). The *Intel 80386 Programmer's Reference Manual* is not absolutely necessary, but every programmer should have a copy. (Intel's phone number is 1-800-548-4725; the order number is 230985-001.)

Coding Conventions

Coding conventions must be defined prior to any serious discussion of a programming language, and perhaps especially assembly language. They are a set of guidelines for program layout and style—the look and feel of a program, if you will. Whereas one could choose endless ways to comment or otherwise express a program, I shall select a single format and express all programs in this manner.

Coding conventions save time, because many of the arbitrary decisions affecting program layout are specified by the convention. Program quality improves, because you are "required" to perform important tasks that might otherwise be neglected. And programs will have a more uniform look, which is of great benefit when more than one programmer works on a project.

The value of uniform program appearance should not be understated, and, while my personal conventions may not be a perfect solution, you should have some valid reason for the choices you make. If a particular style seems so hideous as to induce nausea, you should probably use something else.

Hello World

The trite "hello world" program serves as an introduction to general layout (see page 59). The first two lines of a program contain **title** and **page** directives, which enhance listfile appearance. I prefer to print code in listing form, even when only a few routines are required. The date stamp and code generation are quite valuable, and the time required to edit a listing down to size is minimal.

```
        title   hello world
        page    64,128

        dosseg
        .model  small
        .stack

        .const
hello_text      db  'hello world$'

        .code

;;      main
;
main proc
        mov     ax,@data        ; set data segment to DGROUP
        mov     ds,ax

        mov     ah,9            ; display *hello world*
        lea     dx,hello_text
        int     21h

        mov     ax,4C00h        ; exit program
        int     21h
main endp

        end     main
```

OPTASM, TASM, and recent versions of MASM (5.0 and later) provide simplified segment directives, which offer a straightforward syntax for the otherwise difficult segment specifications. The shorter directives are a welcome language addition and are used to the exclusion of the older formats whenever possible.

The **dosseg** directive specifies the order for the various segments in a program; executable code is first, initialized data follows, and uninitialized data is last. Placing the uninitialized data at the end of a program permits the linker to produce smaller EXE files, which load faster and use less disk space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The **.model** directive specifies the size of a program's code and data. The small memory model permits up to 64K of code and 64K of data, which includes static variables and stack space. Unlike a small model C program, assembly language permits access to any amount of dynamic storage, regardless of the declared model.

While none of the five "simplified" memory models is ideal for advanced assembly language development, they are suitable until the topic of segmentation receives a thorough treatment. For now, the small model is the most efficient and flexible model for our programs, and most examples in this article will use this model or one derived from it.

All memory models support a DGROUP data area consisting of four segments: _DATA for initialized static storage, CONST for initialized constant storage, _BSS for uninitialized static storage, and STACK. The description of DGROUP remains purposely vague until a more detailed explanation of segments, groups, classes, and assumes is complete.

The "hello world" program does not use _DATA storage, but the concept is easy enough. C uses this segment for initialized static variables, as in the following example:

```
int count = 0;
char c = 'a';
```

which is equivalent to:

```
.data
count   dw  0
c       db  'a'
```

The CONST segment is intended for data that should never change. These are almost always ASCII strings, but tables containing prime numbers or other data are also possible. One might think that C compilers would use the CONST segment for static variables declared with the **const** keyword; however; I found this to be untrue.

The DOS program loader makes no attempt to preset storage within the _BSS segment, so whatever values happen to be at those locations in memory determine the initial values for _BSS variables. Needless to say, a correctly functioning program will always write to a location in _BSS prior to reading it. To define _BSS storage, use the **.data?** directive.

```
.data?
arg_count   dw  ?
key_char    db  ?
```

The STACK segment contains the processor stack. (If you do not know what a *stack* is, learn the computer language FORTH, after which you will be an expert.) Like the _BSS segment, STACK is uninitialized storage. The **.stack** directive allocates a program stack, which defaults to 1024 bytes in the absense of an argument.

This directive should be the only line necessary to set up a stack, but because of a feature (that is, a bug) in Microsoft Link, the SS and SP values in the EXE header are set relative to the STACK segment instead of DGROUP. A few lines of code can correct the problem; however, the hello and today programs are trivial and do not require the fix.

```
mov  ax,@data     ; set DGROUP relative stack
mov  ss,ax
sp,offset stack
```

The **.code** directive signals the start of the code segment (named _TEXT, for reasons unknown). This program contains only one procedure, **main**, which is simple enough. The only tricky lines within **main** are the two that select DGROUP as our data segment. The DS register must be set prior to attempts to access memory within DGROUP, because a program always begins with DS equal to some other value.

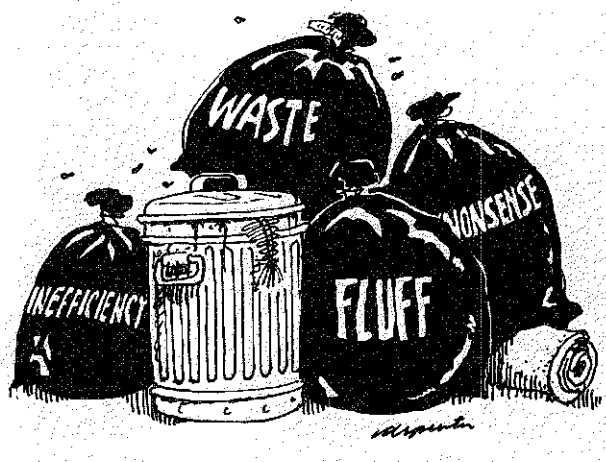
The choice of the name **main** as the program entry point is prudent, given the popularity of the C language. We could expect that a programmer familiar with C would know **main** is the entry point without having to reference the **end** directive. Taking advantage of such knowledge is an important aspect of our effort to program efficiently.

Procedures are defined using **proc** and **endp** directives and not with colons because the directives provide useful structure to the code and permit the use of local labels, which are incredibly handy. The **proc** and **endp** directives also provide good search targets to get to the beginning or end of a long procedure within an editor.

```
main:   mov  ax,@data   ; ** WRONG WAY TO
        mov  ds,ax      ; ** DEFINE A PROCEDURE
        ...
```

Procedure Comments

The two comment lines before the procedure definition demonstrate the simplest form of a procedure comment. While this example provides only a visual aid, the expanded form documents essential functions performed by the procedure.



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Procedure comments always begin with a double semicolon followed by a tab and the procedure name. By convention, the double semicolon is exclusively reserved for this purpose, and every procedure must have such a comment, even if it is as simple as in this example:

```
;;      main
;
```

You can take advantage of this syntax within an editor to locate a procedure definition by searching for the double semicolon, tab, and procedure name. The incremental search feature of an EMACS-like editor can use double semicolons to advance forward and backward through procedure definitions with a single keystroke.

The most important purpose of the procedure comment is to provide documentation on how the procedure affects the processor registers. In the example below, the procedure body consists of a single instruction; therefore, the comments describe the effects of that instruction.

```
;;      divide by cx
;
;       entry   DX AX   dividend
;               CX      divisor
;       exit    AX      quotient
;               DX      remainder
;
divide_by_cx proc
        div     ax,cx
        ret
divide_by_cx endp
```

Two important terms used in conjunction with procedure comments are *entry conditions* and *exit conditions*. Entry conditions are the registers that contain useful values read by the procedure, and exit conditions are the registers that contain results returned by the procedure. Thus, the entry conditions for **divide_by_cx** are the DX AX register pair, which contains the dividend, and the CX register, which contains the divisor. The exit conditions are the quotient in AX and the remainder in DX. CX is unaffected by the routine and is not listed in the exit conditions.

I admit that *entry conditions* and *exit conditions* are not perfect terms, but they describe a very meaningful way to conceptualize procedures—as "black boxes," similar to individual instructions, in which certain values enter and other values leave. The component instructions of a procedure are important only during detailed study or modification of a procedure's behavior.

Continuing the C analogy, entry conditions are similar to function parameters; that is, they are the primary method used to pass information into a function. Exit conditions are similar to function results, with the exception that while C functions permit the return of only a single result, assembly procedures can conveniently return several.

You may notice that the divide instruction's effects on the processor flags are not documented. In this case, one should presume that the flag values are undefined. The **isdigit** procedure provides an example of how to document a returned flag:

```
;;      isdigit
;
;       entry   AL      character
;       exit    Zf      if ascii 0..9
;
isdigit proc
        cmp     al,'0'
        jb      isdl    ; if not digit
        cmp     al,'9'
        ja      isdl    ; if not digit
        cmp     al,al   ; else set Zf
isdl:   ret
isdigit endp
```

Because flags indicate some conditional value, an appropriate way to document them is with some sort of "if ..." phrase. Thus, **isdigit** sets the zero flag if the AL register contains an ASCII digit.

I never use flags as entry conditions, although on occasion I specifically document a routine that has no effect on the flags (see below). Carry and zero are the most common flags used for exit conditions. You will find that carry is the workhorse flag for assembly language programmers because the 80X86 has special instructions that alter this flag.

```
;;      zero all counters
;
;       entry   DS      DGROUP
;       exit    FLAGS   unchanged
;
zero_all_counters proc
        mov     char_count,0
        mov     word_count,0
        mov     line_count,0
        ret
zero_all_counters endp
```

Another important part of the procedure comment is a line that lists which registers are bashed. Consider the following code, which computes string length:

```
;;      strlen
;
;       entry   DS:SI   asciiz string address
;       exit    CX      string length
;       uses    AX
;
strlen proc
        mov     cx,si   ; search for end of string
sl1:    lodsb
        cmp     al,0
        jne     sl1     ;  if not end of string

        dec     si      ;  compute string byte count
        xchg    cx,si
        sub     cx,si
        ret
strlen endp
```

This procedure accepts an ASCIIZ string address in DS:SI, returns the number of bytes in CX and, in the process, uses register AL. A caller to **strlen** cannot expect an important value in the AX register to remain unchanged. (The programmer could save the AX register with **push** and **pop** instructions; however, this is inefficient and almost never necessary with AX. A general guideline for register usage is to trash registers as necessary and save them as required when you write the calling procedure.)

You will find that procedure comments are an extremely valuable form of documentation and the only comments that a programmer should force himself to write while programming (and not when the program is complete). In a future column, I will introduce a program that automatically extracts procedure comments from multiple source files and generates a document on the routines composing a program or library.

By the way, I do not include static variables in either the entry or exit conditions of a procedure comment, although it seems like a good idea. My actions are based more on habit

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and aesthetics rather then the usefulness of the documentation. Try it both ways, and make a choice based on your own experience.

Having introduced procedure comments, we can descend to the next level: the block comment. Every procedure is composed of one or more "functional blocks" of instructions. These blocks perform some simple action and consist of approximately two to eight lines of code. Exactly how a procedure divides into blocks is at the programmer's discretion.

The **strlen** procedure contains two functional blocks of instructions separated by a blank line. The first block copies the initial string address to CX and scans for a zero byte, which indicates end of string. The second block restores the original value of SI and computes the string byte count by subtracting the start address from the end address.

The first line of a block may contain an in-line comment that describes the function of the entire block. All in-line comments should begin in the same column when possible (I use column forty). Avoid unnecessary capitalization or punctuation within comments except to increase clarity.

Long comments can overflow to a second or third line. An additional space after the semicolon indicates continuation of a comment:

```
mov   cx,80 ; set CX to number of 80 char
div   cx    ;  lines in this buffer
mov   cx,ax
```

Within a block, individual lines of code usually provide enough information so that their underlying algorithm is more or less obvious. Exceptions are conditional branches, which frequently benefit from in-line comments describing the conditions of the branch.

As a rule, a programmer should always try to document conditional branches with a comment of the form "; if (*situation*)" where "(*situation*)" describes the condition for which the jump is effective. A second space after the semicolon offers a visual clue that the comment is for the single line of code and not the block. Be sure to avoid useless comments such as "if zero" or "if not equal."

```
jcxz  abc1        ; if no lines
cmp   cx,MAX_LINES
ja    abc2        ; if too many lines
```

Comments are applied to individual lines when the programmer feels that a subtle or otherwise non-obvious aspect of the code requires documentation. As with conditional branch comments, use two spaces between the semicolon and the first word. Occasionally, I enclose such a comment within parentheses to emphasize that the comment is a note and not a description of an algorithm.

```
     mov   dx,3F8h  ; wait for serial char
spl: in    al,dx
     test  al,1     ;  (RDA bit for 8250 is 1)
     jz    spl      ;  if no receiver data
```

Just as a particular line of code may not require a comment, neither may a particular block. This is the programmer's decision. For instance, I would not include any in-line comments with the **strlen** procedure were it not an example.

Do avoid the practice of commenting every line of code. A majority of such comments will simply restate the obvious function of the code. The resulting volume of comments not only distracts the reader but also obscures any useful documentation.

```
mov  al,[si]  ; load AL from memory ** TOTALLY
cmp  al,100   ; compare AL with 100 ** USELESS
je   bad1     ; jump if equal       ** COMMENTS
```

Procedure Names

Names for three of the previous routines (**main**, **isdigit**, **strlen**) are identical to corresponding C functions; this has valuable self-documenting characteristics. Were it not for this, such terse names would usually be poor choices for procedures in a program.

Creating a good procedure name is a frustrating process. The objective is to come up with a name that is easy to remember and descriptive. In order to restrict the set of possible names, I use a formula for creating procedure names, which is to make them three words long and usually of the form ***verb adjective noun***. Examples include **read_next_char** and **show_one_line**.

A slightly different form is used for a collection of routines: ***group verb noun***. For example, **ascii_open_file** and **ascii_read_line** might be routines for manipulating ASCII files. Data objects are described with two-word names—for example, *word_count* and *member_name*.

Some of the benefits of these methods are:

- With fewer choices, the programmer saves time creating a name.
- Names are easier to guess.
- The programmer knows whether a name refers to code or data outside of any context.

The quality of your choice becomes evident six months later: If you can guess the correct name, it's good; if someone else can guess the name, it's great! Remember, the three-word rule is a guideline and not a law. If you feel that a two- or four-word name is required, use it.

Local Labels

Within a procedure, the looping and jumping instructions require some sort of symbolic specification for the target of their branch. A common approach is to choose some long name descriptive of the branch condition, such as **loop_here_more_string**. This method is poor for several reasons:

- More than one conditional branch may reference the label.
- The label occupies a valuable line within the editor.
- The label may conflict with another in a different procedure.
- Habit may force you to choose a misleading label.

A much better approach is to use dummy labels within a procedure. OPTASM, TASM, and MASM (version 5.1) provide for "local labels" within a procedure. Unfortunately, they all implement incompatible schemes; so, we will have to resort to a method that works with any assembler.

Choose some unique two- or three-character abbreviation and add a number as a suffix. This works well with the three-word convention for procedure names, because initials are usually unique. A procedure named **read_next_char** would use local labels: rnc1, rnc2, and so on. With this method, you will waste less time creating labels, have more attractive

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Commenting Disassembler!

SOURCER

- **SEE HOW PROGRAMS WORK**
- **EASILY MODIFY PROGRAMS**

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80286/287 and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

On my list of programs that I simply won't do without!
—Robert Hummel, Senior Technical Editor, PC Magazine, 4/26/88

SAMPLE OUTPUT

Fully automatic

Program header

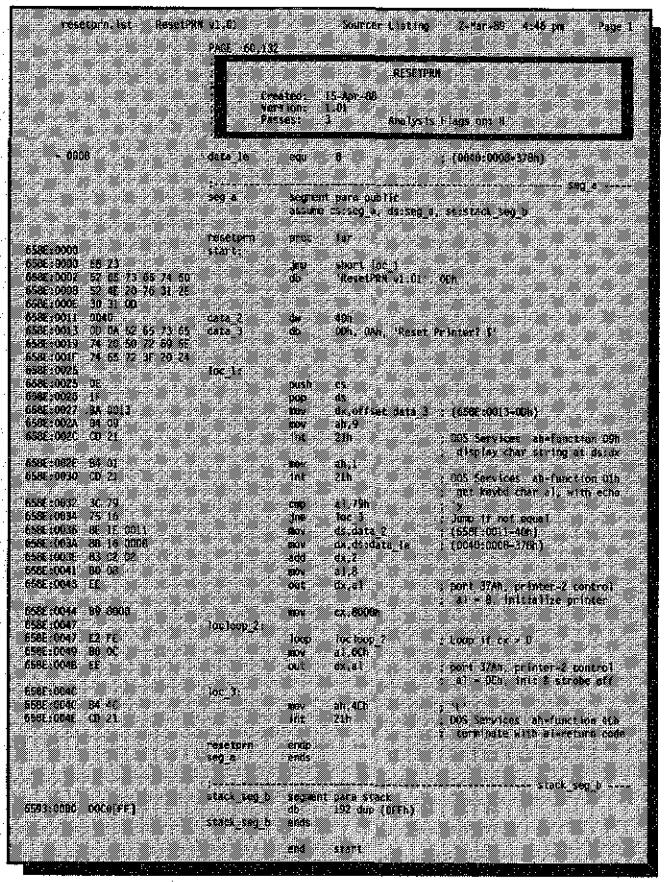
Assembler directives

Determines data areas and type

Detailed comments

Simulator follows segment changes

Easy to read format


resetprn.lst ResetPRN v1.01 Source Listing 26-Mar-88 4:48 pm Page 1
PAGE 60,132
RESETPRN
Created: 15-Apr-88
Version: 1.01
Passes: 3 Analysis Flags on R
data_1e equ 0 ; (0040:0000+378h)
seg_a segment para public
assume cs:seg_a, ds:seg_a, ss:stack_seg_b
resetprn proc far
start:
jmp short loc_1
db 'ResetPRN v1.01', 0Ch
data_2 dw 40h
data_3 db 0Dh, 0Ah, 'Reset Printer! $'
loc_1:
push cs
pop ds
mov dx,offset data_3 ; (658E:0013+0Dh)
mov ah,9
int 21h ; DOS Services ah=function 09h
; display char string at ds:dx
mov ah,1
int 21h ; DOS Services ah=function 01h
; get keyed char al, with echo
cmp al,79h ;
jne loc_3 ; Jump if not equal
mov ds:data_2,ax ; (658E:0013+40h)
mov ax,ds:data_1e ; (0040:0000+378h)
add dx,2
mov al,8
out dx,al ; port 37Ah, printer-2 control
; al = 8, initialize printer
mov cx,8000h
loc_2:
loop loc_2 ; Loop if cx > 0
mov al,0Ch
out dx,al ; port 37Ah, printer-2 control
; al = 0Ch, init & strobe off
loc_3:
mov ah,4Ch ; 'L'
int 21h ; DOS Services ah=function 4Ch
; terminate with al=return code
resetprn endp
seg_a ends
stack_seg_b segment para stack
db 192 dup (0CFh)
stack_seg_b ends
end start

(Source code output and inline cross reference can also be selected)

BIOS SOURCE

- **CHANGE AND ADD FEATURES**
- **CLARIFY INTERFACES**

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$99.95	**UNPACKER**™ Unpack packed EXE files and more	$39.95
SOURCER with BIOS Pre-Processor	139.95	**ASMtool**™ Assembly source analyzer and flowcharter	89.95

USA Shipping & Handling $3; Outside USA $15; CA Res. add sales tax; PS/2, AT, XT, and PC are trademarks of IBM Corp.

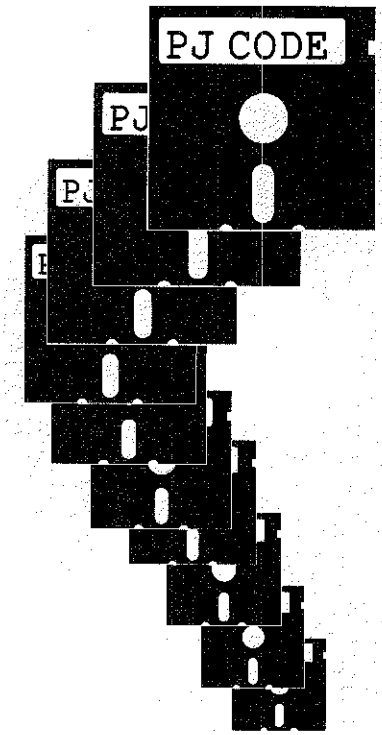
All our products come with a 30 day money back satisfaction guarantee. Not copy protected. To order or receive additional information just call!

VISA **1-800-662-8266** MasterCard

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 905, Dept. J3, San Jose, CA 95128 (408) 296-4224

Number **125** on the Reader Service Card

PJ Code Is Available!



On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
800-234-0386
503-747-0800 in Oregon

All Code Disks $10.00 each.
To order see ad on page 88.

PJ Code DiskScriptions
are available for only $39.95.
To order see ad on page 8.

On BBS from:

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

FIDO/OPUS Network
Blazer Net 152 Node 9
503-343-9724

Generation Five BBS
(Voice) 301-495-6690
(Data) 301-495-2932
Accessible via PC Pursuit, Washington D.C.
From Files submenu, select "P" for Personal Downloads.

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Code from "A Line of Code" by John Otken

1

```
title   today - print filenames with today's date
        page    64,128

ms_dos          equ     21h     ; dos software interrupt vector

find_first struc                ; for find_first & find_next DOS functions
  ff_reserved   db  21 dup (?)  ; reserved for dos
  ff_attribute  db  ?           ; file attribute byte
  ff_time       dw  ?           ; file time of last write or update
  ff_date       dw  ?           ; file date of last write or update
  ff_size       dd  ?           ; file byte count
  ffname        db  13 dup(?)   ; file name
find_first ends

        dosseg
        .model small
        .stack

        .const
star_dot_star   db      '*.*',0

        .data?
since_date      dw      ?       ; today's date in two byte format
since_time      dw      ?

dta     find_first      <>      ; allocate structure for find_first

        .code

;;      get dos date
;
;       exit    AX      two byte date
;       uses    CX,DX
;
; <         high          > <          low          >
; 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
;  y  y  y  y  y  y  y  m  m  m  m  d  d  d  d  d
;
get_dos_date proc
        mov     ah,2Ah
        int     ms_dos

        mov     ax,cx                   ; set year
        sub     ax,1980
        and     ax,1111111b
        mov     cl,9
        shl     ax,cl

        and     dl,11111b               ; set day
        or      al,dl

        and     dx,111100000000b        ; set month
        mov     cl,3
        shr     dx,cl
        or      ax,dx
        ret
get_dos_date endp


;;      main
;
main    proc
        cld
        mov     ax,@data                ; our data segment is DGROUP
        mov     ds,ax

        mov     ah,1Ah                  ; set data transfer address for
        lea     dx,dta                  ;  find first and find next
        int     ms_dos                  ;  dos functions

        call    get_dos_date            ; match files with today's date
        mov     since_date,ax
        mov     since_time,0            ;  and any time on or after midnight
```

UNIQUE PREFIX FOR MEMBER NAMES

ASCIIZ STRING CONSTANT

UNINITIALIZED STATIC VARIABLES

ALPHABETIZED PROCEDURES

BEGIN EXECUTION AT MAIN

MULTI-LINE BLOCK COMMENT

2

```
        lea     dx,star_dot_star        ; use "*.*" file specification

mai1:   mov     ah,4Eh                  ; find first file in directory
        mov     cx,0                    ;  (zero attribute for normal files)
        int     ms_dos
        jc      mai4                    ;  if no files or other dos error

mai2:   mov     ax,dta.ff_date          ; check file date and time
        cmp     ax,since_date
        jb      mai3                    ;  if date too early

        mov     ax,dta.ff_time
        cmp     ax,since_time
        jb      mai3                    ;  if time too early

        lea     si,dta.ff_name          ;  else display matching file name
        call    outchr_asciiz
        call    outchr_newline

mai3:   mov     ah,4Fh                  ; find next file
        int     ms_dos
        jnc     mai2                    ;  if more files in directory

mai4:   mov     ax,4C00h                ;  else exit program
        int     ms_dos
main    endp


;;      outchr
;
;       entry   AL      char
;
outchr  proc
        push    ax
        push    dx
        mov     ah,2
        mov     dl,al
        int     ms_dos
        pop     dx
        pop     ax
        ret
outchr  endp


;;      outchr asciiz
;
;       entry   DS:SI   asciiz string
;       uses    AX,SI
;
outchr_asciiz proc
oa1:    lodsb
        cmp     al,0
        je      oa2
        call    outchr
        jmp     oa1
oa2:    ret
outchr_asciiz endp


;;      outchr newline
;
;       uses    AX
;
outchr_newline proc
        mov     al,'M'-'@'
        call    outchr
        mov     al,'J'-'@'
        call    outchr
        ret
outchr_newline endp

        end     main
```

NOTE COMMENT WITHIN PARENS

CONDITIONAL BRANCH COMMENT

DUMMY LOCAL LABEL

THESE ROUTINES ARE SO TRIVIAL THEY DO NOT REQUIRE COMMENTS.

THEY ARE ALSO GENERALLY USEFUL AND WILL BE PART OF A LIBRARY.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

code, and see more lines of code within your editor. In addition, you can instantly know whether a symbol is a local label or a procedure name. The downside is that you will, on occasion, have to resequence your labels.

TODAY.ASM

TODAY is a program that displays a list of files created or modified since midnight. It is useful for locating the names of "active" files within a large directory. The program includes an equate and structure definition, positioned toward the beginning of the source file before any data or code-generation statements. Notice that the structure members are prefixed with "ff_". Structure members are frequently used in such a way that their parent structure is not obvious. A unique prefix offers a clue to the parent and helps prevent the use of members from the wrong structure.

The procedures within TODAY are alphabetized. This increases the uniform layout of source files, which is so important for multiprogrammer projects. Organizing your procedures this way will also make listings more usable, since locating a procedure will not require a linear search.

Summing Up

Good coding conventions make a tremendous difference in the programming effort of a large project. This is true for any language. Often, the conventions restrict the possible variations in a program so that the programmer spends more time designing data structures and algorithms and less time making pointless decisions. Remember the following concepts covered in this article when you begin your next project:

- Attractive programs are easier to write and maintain.
- Use title and page to enhance program listings. If you are going to make a hard copy of your program, use the listing form.
- Use the simplified segment directives (**dosseg**, **.model**, **.data**, **.code**, etc.) instead of the older forms.
- Borrow useful conventions from other languages; for example, the use of "main" as the entry point for a program.
- Define procedures using **proc** and **endp** directives.
- Use procedure comments to document entry and exit conditions.
- Separate functional blocks of code with a blank line and use an in-line comment to describe the function of the block.
- Document conditional branches with an in-line "if" comment.
- Do not comment every line of code.
- Choose three-word procedure names and two-word data names.
- Within a procedure, form local labels from the procedure initials or use the local label feature of your assembler.
- Use alphabetical order for the procedures within a source file to enhance listings. PJ

John Otken is the president of Soft Advances, where he develops debuggers for PCs and embedded systems applications. He may be reached at 10811 Washington Blvd., Suite 205, Culver City, CA 90232, 213-559-7015.

The Indispensible Quick Reference Guide for Assembly Language Programming

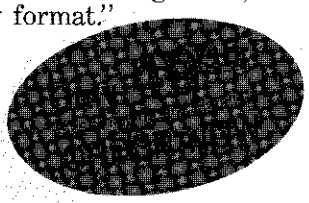
MASM QuickRef

- MASM Instruction Set
- Video Interrupts
- Keyboard Scan Codes
- Display Attributes
- Shift Mask Codes
- Hex-Dec Binary Conversion

Now Also Includes Programmer's Calculator

Memory Resident — Pop-Up Windows

"It's a friendly tool that lets you quickly find commands and procedures in an organized, concise pop-up window format."



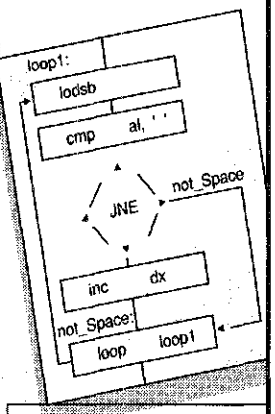
Ultra-X, Inc.
2005 De La Cruz Blvd.
Suite 115
Santa Clara, CA 95050
(408) 988-4721

Number **290** on the Reader Service Card

ASMFLOW $99.95

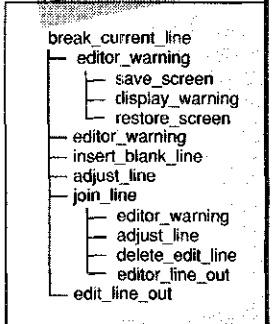
AT LAST!

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE



FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80386/387
- Context-Sensitive Help
- Menu/Batch/Command line Operation
- MASM 5.1 Compatible



QUANTUM SOFTWARE

19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826 – Free Demo

VISA • MC 30 - Day Money Back Guarantee

Number **252** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Fast String Searching

by

Adrian B. Crum

This article describes a little assembly language routine that I use to search for a string within text. It is based upon an algorithm that was discovered by R. S. Boyer and J. S. Moore. A description of this algorithm can be found in a book entitled *Algorithms*, by Robert Sedgewick [Addison-Wesley, 1983 (revised 1989)].

The algorithm implements right-to-left scanning, which means that the text is operated upon from left to right but the pattern to search for is compared with the text from right to left. A table constructed in memory is used to determine what to do when a mismatch occurs.

To demonstrate, let's search for the pattern *OUR* in the text *PROGRAMMER'S JOURNAL*, leaving out the apostrophe for simplicity's sake (see the figure). Proceeding from right to left to match the pattern, we first compare the *R* in the pattern to the first *O* in the text. These don't match. Also notice that the *O* can be found in the pattern. This is where we use the table, which tells us how far to shift the pattern to the right for each character in the alphabet if that character causes a mismatch in the text. For the pattern *OUR*, the table would contain the number 3 for every letter in the alphabet except *O* and *U*. *O* would have the number 2, and *U* would have 1. In this comparison, the table would return 2. We shift *OUR* over two characters so that the two *O*s line up. Now the *R* in the pattern is compared to the second *R* in the text. Here we have a match, so we compare the rest of the pattern to the text (the table is not used). Since these don't match, we can shift the pattern over three more characters. Next, we compare the *R* in the pattern to the second *M* in the text. These don't match, and the table tells us to shift the pattern over three more characters. This process continues until the *R* in the pattern is compared to the *O* in JOURNAL. The pattern shifts over two characters, and the match is found.

This took only nine comparisons. If we had instead scanned the text for all occurrences of *O*, then 17 comparisons would have been made. As you can see, this algorithm is quite efficient.

The problem with implementing this algorithm in assembly language is constructing the "shift table." This table must be initialized for every search to be performed (since every pattern may be different), which leads to (in my opinion) excessive code. Fortunately, the 8088 has an **SCASB** instruction that can eliminate the shift table. If we use **SCASB** to scan for the mismatched character in the pattern, CX will return a value that we can use instead of the value returned by the shift table.

Listing 1 is my implementation of the Boyer–Moore algorithm. Up to 64K of text can be searched with a single call to this routine. I have preserved some of the registers so that repeated calls may be made to the routine in the event of multiple matches. PJ

Adrian B. Crum is a computer hobbyist and part-time consultant. He has been programming computers since 1978. Adrian can be reached at P.O. Box 3404, Fontana, CA 92334, 714-350-8135.

Code follows.

```
Text: PROGRAMMERS JOURNAL
   1. OUR                    'O' IN TABLE SAYS SHIFT 2
   2.   OUR                  CHAR-BY-CHAR COMPARISON = NO MATCH, SHIFT 3
   3.     OUR                'M' IN TABLE SAYS SHIFT 3
   4.       OUR              'S' IN TABLE SAYS SHIFT 3
   5.         OUR            'O' IN TABLE SAYS SHIFT 2
   6.           OUR          CHAR-BY-CHAR COMPARISON = MATCH FOUND
```

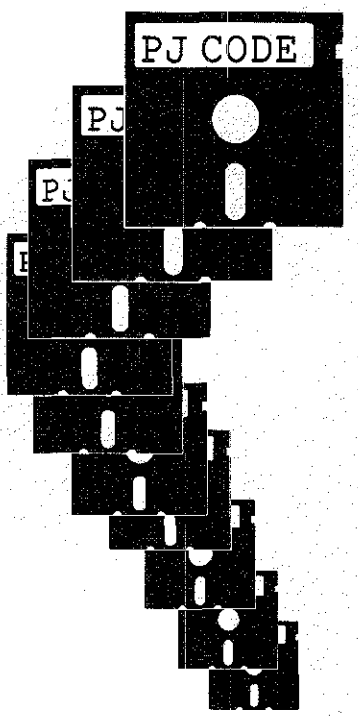

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Fast String Searching" by Adrian Crum

```
; SEARCH.ASM    A STRING SEARCH ROUTINE
; (C) 1987 BY ADRIAN CRUM
;
; IMPLEMENTS A MODIFIED BOYER-MOORE ALGORITHM
; THE 'NEXT TABLE' IS REPLACED BY THE 8086 'SCASB' CAPABILITY
;
; CALLING PARAMETERS:
;
; BX = LENGTH OF PATTERN TO SEARCH FOR
; DX = OFFSET OF LAST BYTE OF TEXT TO BE SEARCHED
; SI = OFFSET OF FIRST BYTE OF TEXT TO BE SEARCHED
; DI = OFFSET OF PATTERN TO SEARCH FOR
; ES: = DS:
;
; RETURNS CARRY CLEAR AND SI=MATCHING TEXT OR CARRY SET IF NO MATCH.

SEARCH  PROC    NEAR

        DEC     BX              ; BX = pattern length - 1
        MOV     AL,[BX+DI]      ; Get the last char in pattern
        MOV     AH,AL           ; Save the char in AH
        CLD

COMPARE:
        CMP     SI,DX           ; Reached end of text yet?
        JA      NOT_FOUND       ; Yes, bail out
        MOV     CX,BX           ; Put pattern length in CX
        INC     CX
        CMP     AL,[BX+SI]      ; Last char in pattern match?
        JNE     SHIFT           ; No, shift pattern
        PUSH    SI              ; Save the pointers
        PUSH    DI              ; Compare pattern with
        REPZ    CMPSB           ; this portion of text
        POP     DI              ; Restore pointers
        POP     SI
        JE      FOUND           ; Match found, bail out
        MOV     CX,BX
        INC     CX              ; CX = pattern length

SHIFT:  PUSH    DI              ; Save the pointer
        MOV     AL,[BX+SI]      ; Get non-matching char
        MOV     BX,CX           ; Set up BX for possible jmp to SHIFT1
        REPNE   SCASB           ; Non-matching char in pattern?
        JCXZ    SHIFT1          ; No, proceed
        ADD     SI,CX           ; Yes, shift pattern (CX) chars
        JMP     SHORT   SHIFT2
SHIFT1: ADD     SI,BX           ; Shift pattern (pattern len) chars
SHIFT2: POP     DI              ; Restore pointer
        MOV     AL,AH           ; Restore AL
        JC      NOT_FOUND       ; If segment wrap, bail out
        DEC     BX              ; BX = pattern length - 1
        JMP     COMPARE         ; Loop

FOUND:  INC     BX
        CLC                     ; Search successful
        RET
NOT_FOUND:
        STC                     ; Search unsuccessful
        RET

SEARCH  ENDP
```


PJ Code Is Available!



On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
800-234-0386
503-747-0800 in Oregon

All Code Disks $10.00 each.
To order see ad on page 88.

PJ Code DiskScriptions
are available for only $39.95.
To order see ad on page 8.

On BBS from:

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

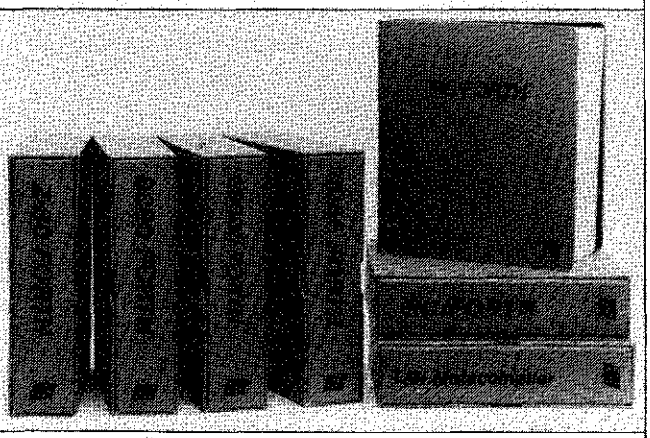
FIDO/OPUS Network
Blazer Net 152 Node 9
503-343-9724

Generation Five BBS
(Voice) 301-495-6690
(Data) 301-495-2932
Accessible via PC Pursuit, Washington D.C.
From Files submenu, select "P" for Personal Downloads.

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170

PRODUCED 2006 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TOTAL CONTROL
with *LMI FORTH*™



For Programming Professionals:
an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers
- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler
- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler
- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:
- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Number **63** on the Reader Service Card

The C Programmer's Companion
Now Available for OS/2 !!!

CCSYM - C Symbol Compiler
* Analyzes ANSI C source code. Microsoft/Turbo extensions supported.
* Automatically generates separate commented function prototype files for all global and local (static) functions.
* Log File lists all your symbol definitions with detailed usage statistics.
* Generates a "symbol database" which CCRIDER uses to quickly retrieve ANY symbol definition in your application, no matter how large it is.
* Incremental compilation supports efficient database updates.

CCRIDER - Pop-up Symbol Lookup
* Pop-up utility which converts your own editor into a powerful hypertext programming environment.
* Point at a symbol in your editor, then pop-up its definition in a window.
* One keystroke "Edit" command allows you to edit the definition in your own editor, no matter which file or directory it was defined in.
* "Paste" command inserts the definition into your currently edited file.
* Supports multiple editing sessions under OS/2.

DOS Version $89
OS/2 Version $179

FREE DEMO

Western Wares
Box C Norwood, CO 81423
(303) 327-4898

Number **100** on the Reader Service Card

The Programmer's Productivity Pack

THE MOST USEFUL PROGRAMMER'S UTILITIES TOGETHER IN ONE INTEGRATED PACKAGE!

The Programmer's Calculator
• Decimal, Binary, Octal and Hexadecimal simultaneously • Signed and Unsigned modes • SHL/SHR, SAL/SAR, ROL/ROR, RCL/RCR • AND, NOT, OR, XOR, XCHG • Memory Slots • and more!

The ASCII and Color Attribute Chart
• Displays and describes color and monochrome attributes • Enlarged display of box drawing and graphics characters • Entire ASCII and IBM extended character set!

The Keystroke Reference Center
• BIOS Scan Codes and dBASE® INKEY() values for all keys and key combinations • Portability Notes for keyboard differences • Live display of the BIOS Keyboard Status Bytes • and more!

Keystroke Recording and Playback
• Up to 100 recordings of 1,000 keystrokes each • Save recordings to disk files • Grab text from screen for use as a Keystroke Recording • and more!

plus! Screen Capture, Screen Blanking, Background Printing, Hard Disk Parking!

plus!! Memory-resident & Multitasking Versions!

plus!!! 26"x20" ASCII Poster (posters also available separately)

ONLY $79.95
plus Shipping and Handling
30 Day Money Back Guarantee

5322 Rockwood Ct. El Paso, Texas 79932

EASY TO LEARN, EASY TO USE
(915) 584-7670
Free Brochure Available

FALK DATA SYSTEMS

Number **267** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Database Normalization

by

Bill Madison

Structured analysis and design techniques have led to greatly simplified implementation, debugging, and maintenance of programs. In a similar vein, *database normalization* can lead to reduced data volume, simplified retrieval, and simpler (and better controlled) updating procedures. Database normalization is a body of techniques that optimizes the file and record structures of databases for storage size, query and update simplicity, and data redundancy.

But good methods can become bad dogma. Any database can (and probably should) be run through the full gamut of normalization exercises. But occasionally, because of peculiarities of the manner in which the database is accessed, something less than full normalization is desirable. The fully normalized database needs to be "detuned" to achieve specific parochial ends. Thus, just as with any power tool, knowledge of when *not* to use it (or when to *stop* using it) is just as important as knowing when to use it.

Data Models

Ask a roomful of skilled microcomputer software developers to define *database*. You will probably get as many definitions as there are people in the room. And the definitions may have little in common. For this article, the definition given by Wiederhold (see references) will be used:

> A [computerized] *database* is a collection of related data. The data storage for a database is accomplished by the use of one or more files. A *file* is defined to be a collection of similar records kept on secondary computer-storage devices. . . . A *record* is defined . . . to be a collection of related *fields* containing elemental data items. . . . A *data item* typically represents a value which is part of a description of an object or an event.

This definition completely avoids any dependence on a formal database management system or accessing method. These omissions are not accidental.

Three data models are in general use in modern database management systems (DBMSs): the *hierarchical*, *network*, and *relational* models. These are "nested models," in the sense that a network model can completely describe a hierarchical model, and a relational model can describe a network model.

A hierarchical data structure is one in which two or more record structures are associated in a "parent–child" relationship. Each instance of a record structure (the "parent") potentially "owns" one or more instances of the same or other record structures (the "children"). These child records may or may not have the same structure as the parent, and any given record can have no more than one parent.

The network structure, by definition, permits records to have multiple parent records in addition to the multiple child records permitted by the hierarchical structure.

Schemata

If the purpose of a database is to serve as a generally available data reservoir, then the data must be readily accessible. The data and the files containing them must be structured and managed so that desired data elements can be retrieved in a consistent manner. The DBMS provides this structuring-and-management facility.

For this to happen, the location of the data within a file and record must be known to the DBMS. Also, the retrieved data must be assembled into logical records usable by the application. A "logical record" is a record seen by the application, which may be made up of fields from several different files. It is constructed by the DBMS at the time data are requested. Its structure and content, therefore, will not necessarily reflect the structure and content of any of the "physical records" that exist on the storage medium. The collection of these logical records is known as the application's *view* of the database.

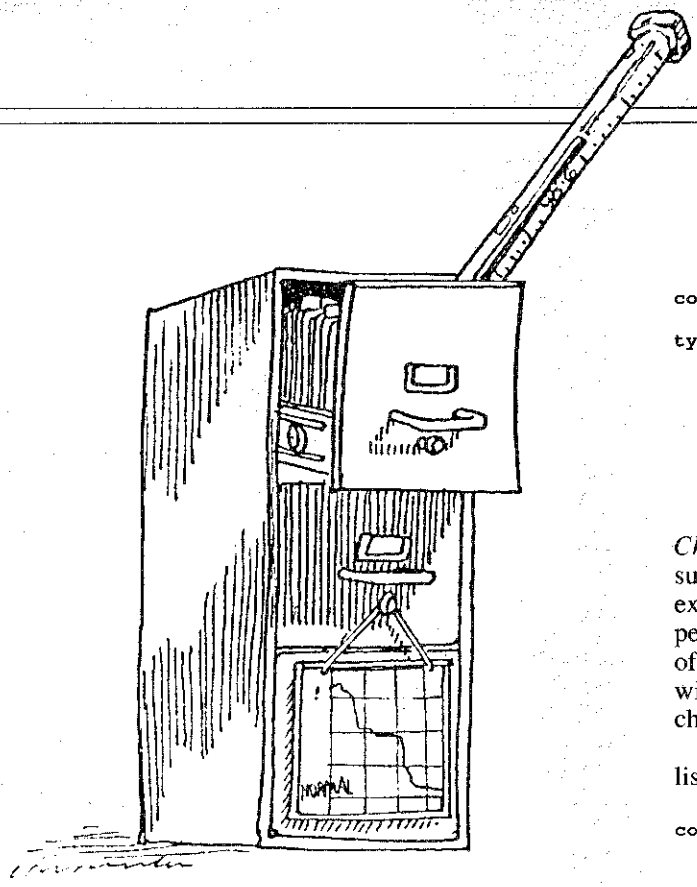
Also, in that "best of all possible worlds," all aspects of the underlying (physical) file and record structures should be hidden from the application. In particular, the mapping rules should not be the concern of the application. The so-called SPARC Three-Schema Model, proposed in 1978 by ANSI/X3/SPARC, provides a framework for alleviating this problem (see sidebar).

Relations

It is obvious that a network data model is a generalization of a hierarchical data model. Any hierarchical model can be described by a network model. It is not quite so obvious that a relational model can be used to completely describe a network model (and therefore a hierarchical model). For now, I simply introduce the idea to relieve your anxiety over whether we are losing any generality when we concentrate our discussion on relational models.

A *relation* may be described as a table, or a flat file of fixed record length. Thus, a relation can be thought of as containing column headings, rows, and data entries (one datum per cell—think of a spreadsheet). The column headings are often called *attributes*, similar to a field name. The values in each cell of a column are drawn from the universe of legal

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



values for that attribute, known as the *domain* of the attribute. The collection of values that make up one row of the table (relation) is known as a *tuple* (rhymes with *couple*) and is analogous to a record.

Within a table, records (tuples) may not be repeated; each must be unique. Further, tuples are unordered within a relation (interchanging two records does not change the relation). Similarly, attributes are unordered. Finally, all attribute values must be atomic; that is, structured attributes (arrays, records, and such) are not allowed. A relation (table) that satisfies these four criteria is "normalized," or is in *First Normal Form (1NF).*

First Normal Form

As an example of a nonatomic attribute, consider the following pseudo-PASCAL record structure:

```
const
  MaxKids = 6;
type
  Emp_Record =
    record
      Emp_ID_Num     : longint;  {Employee ID number}
      Employee_Name,
      Spouse_Name    : string;
      Child_Name     : array[1..MaxKids] of string;
      end;
```

Child_Name is a nonatomic attribute. A table (relation) of such records (tuples) is therefore not normalized. Further, except for those cases in which a family has the maximum permitted number of children, space will be wasted because of surplus space in the *Child_Name* array. Finally, problems will arise when that unusual family with more than *MaxKids* children arises.

The record structure can be normalized simply by establishing MaxKids attributes to replace the *Child_Name* array:

```
const
  MaxKids = 6;

type
  Emp_Record =
    record
      Emp_ID_Num     : longint;  {Employee ID number}
      Employee_Name,
      Spouse_Name,
      Child_Name_1,
      Child_Name_2,
      Child_Name_3,
      Child_Name_4,
      Child_Name_5,
      Child_Name_6   : string;
      end;
```

This record (tuple) structure is clearly normalized; all attributes are atomic. Further, if we assume that no ranking is implied by the ordering of the *Child_Name_x* attributes, then the attributes within the relation may be assumed to be unordered. Finally, the ordering of tuples within the relation is immaterial. Thus, all of the criteria for normalization (First Normal Form, or 1NF) have been met.

The problem of wasted space owing to unused (empty) attributes within the majority of the records in the table continues. How many families today have six children? The problem of tuple overflow when dealing with unusually large families also is still present.

There is another way to normalize this record. The record could be redefined to contain only one *Child_Name*, and the specific instance of the record could be repeated in the database once for each child in the family, changing only the *Child_Name* field:

```
type
  Emp_Record =
    record
      Emp_ID_Num     : longint;  {Employee ID number}
      Employee_Name,
      Spouse_Name,
      Child_Name     : string;
      end;
```

SPARC Three-Schema Model

1. The **internal schema,** of which there is exactly one, describes the physical file and record structures plus any hardware specifics of the storage. It functionally approximates the CODASYL schema.

2. The **external schema,** of which there can be many, describe the application's view of the data base. The view can change from application to application. This accounts for the possible simultaneous existence of multiple external schemata. The external schema functionally approximates the CODASYL subschema minus the mapping rules.

3. The **conceptual schema,** of which there is exactly one, relates the stored data to the information needs of the enterprise, manages the mappings necessary to retrieve data via the internal schema, and provides a stable set of record structures *onto* which the external schemata will map. The CODASYL model has no functional equivalent to the conceptual schema.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This record structure is also in 1NF. Further, this structure avoids both the unused space problem and the overflow problem mentioned above. In their place, however, is introduced the problem of repeated data: The *Emp_ID_Num*, *Employee_Name*, and *Spouse_Name* fields will be repeated for each child in the family.

It's a judgment call (usually) as to which of these two approaches to normalization is better. Typically, the second will result in easier analysis. More important, this illustrates that there is usually no one "correct" way to place a database into normal form (not only 1NF, but also 2NF–5NF). Each individual will impose different, frequently subjective, criteria in the course of the design. The only criterion of "correctness" that can be applied addresses the question of whether or not there is a more convenient design for query or update.

First Normal Form, as presented above, represents the weakest level of normalization and is presupposed in all discussions of relational databases. Second through fifth normal forms—the real meat of database normalization—deal primarily with the elimination of redundant data between the domains of the various attributes of a database.

To achieve this redundancy elimination, a set of tools must be available that allows us to split relations into two or more new relations and to recombine them later so that other new relations (or the original relation) can be constructed. One such toolkit is the so-called relational algebra (see sidebar).

Keys

The unqualified term *key* is one of those best suited to cause unlimited confusion. We speak of "keyed attribute" and mean one that has been indexed. In the relational literature, we encounter the terms *primary key*, *alternate key*, *candidate key*, and *foreign key*. In other areas of database literature, we encounter *index key*, *search key*, *ordering key*, *secondary key*, . . . ad infinitum, ad nauseam.

The various types of keys referred to in this article will be defined at the time they are introduced. Keys can be either simple or composite. Simple keys involve only a single attribute; composite keys involve more than one, potentially even an entire tuple.

SUP_ID	WAREHOUSE	CITY	PART_NUM
S1	30	Detroit	P1
S1	30	Detroit	P2
S1	30	Detroit	P3
S2	10	Boston	P1
S2	10	Boston	P3
S3	20	Chicago	P4
S3	20	Chicago	P5
S4	10	Boston	P4
S4	10	Boston	P5
S4	10	Boston	P6

Table 1

Second Normal Form

Before proceeding further, we shall define a *nonkey attribute* as any attribute that does not participate in the primary key of the relation concerned. For simplicity, assume in this immediate discussion that any relation being discussed has no alternate keys—keys that could, at the designer's option, serve as the primary key. This restriction is unnecessary, but it does simplify the discussion a bit.

Additionally, we define functional dependence between attributes as implying that, given attributes *A* and *B*, *A* functionally determines *B* if (and only if) for every *A*-value in R, precisely one *B*-value is associated; notationally:

$A \longrightarrow B$.

Also, we define two or more attributes as being mutually independent if none of the attributes concerned is functionally dependent upon any of the others.

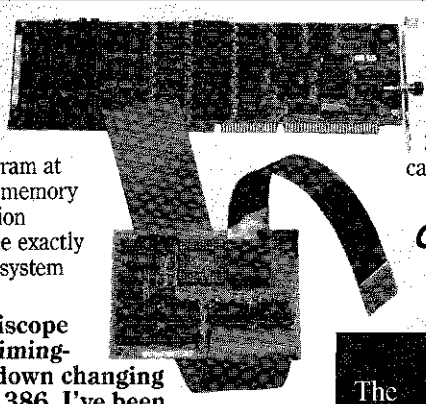
We can now define 2NF as a relation that is in 1NF and in which all nonkey attributes are fully functionally dependent upon the primary key.

Note that a relation R that is in 1NF but not in 2NF can relatively easily be brought into 2NF by *projection* of R into two or more relations, say $R_1, R_2, \ldots R_n$, in which each nonkey attribute of R_i is fully functionally dependent upon the primary key of R_i.

NEW FROM PERISCOPE

Real-Time Debugging For 286s and 386s up to 25MHz

Periscope® IV brings you a new generation of hardware-assisted debugging. It works on virtually any 286 or 386 with an AT-style bus running up to 25MHz with zero wait states. With Periscope IV, you can run your program at full speed while tracking down unwanted memory overwrites. And you can use the information captured in its real-time trace buffer to see exactly what the system is doing, and to improve system performance.



"The hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk® Mk. 4 and there's just no better way to do it," says user Jeff Garbers, Crosstalk Communications' Director of Software Development.

Call now for our FREE four-page Model IV brochure or to talk about your debugging needs. Prices range from $145 for just software to $2,995 for a Model IV for a 25MHz 80386 system. We accept major credit cards and qualified company purchase orders.

Order your Periscope, Toll-Free, Today! 800-722-7006

The Periscope Company, Inc.

1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404/875-8080
FAX 404/872-1973

Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

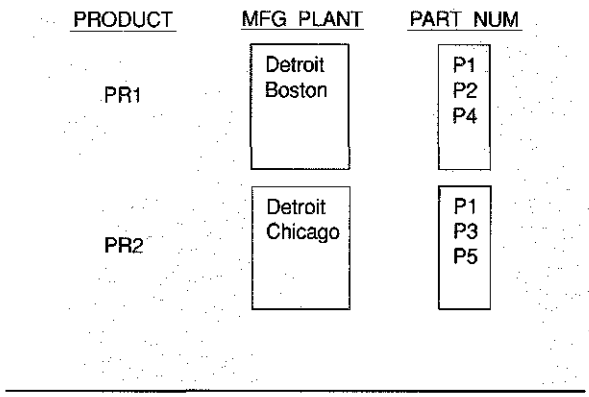


Figure 1

For an example, let's consider a PARTS SUPPLIER database for a manufacturer in some industry. The 1NF relation might look like Table 1 (which is somewhat abbreviated—I've left out fields like supplier name and address). We'll assume that whatever city the supplier is in, he or she will ship to the nearest warehouse. Thus, any supplier city will always correspond to some fixed warehouse. Several cities may be served by the same warehouse, but the converse is not true.

The key here is composite; specifically, (SUP_ID + PART_NUM). But WAREHOUSE and CITY are not functionally dependent on the key; they are dependent only on the SUP_ID and are independent of PART_NUM.

This relation is in 1NF, since all of the attributes are atomic. But consider the complexity of the update problem. For example, were supplier S1 to relocate to Chicago, it would be necessary to search the entire database for each instance of S1 and change both the CITY and the WAREHOUSE fields. However, this relation can be decomposed into tables 2 and 3.

A *join* operation using SUP_ID as the common attribute will exactly reproduce the original relation (that is, we have achieved a *lossless* decomposition), but the update problem mentioned above has been eliminated. And, if you'll take my word for it, the relation is now in 2NF.

Problems still exist, however. Consider the effect of building a new warehouse in a location that might cause a realignment of the CITY–WAREHOUSE correspondence. Without going into excruciating detail, visualize the effect this might have on the relation in Table 2. That relation can also be

The Relational Algebra

In the following discussion, *union compatibility* between relations implies that the two relations are of the same order (i.e., they have the same number of attributes) and that corresponding attributes are based on the same domain.

Given a set of relations comprising a data base, these relations can be manipulated with the following operators to form new relations:

SELECT Builds a new relation by extracting specified tuples (rows) from a specified relation (table).

PROJECT Builds a new relation by extracting specified attributes (columns) from a specified relation (table).

PRODUCT (also known as **OUTER JOIN**) Builds a relation from two specified relations. The tuples of this new relation are all possible concatenations of tuples from the parent relations, taking one from each of the parents.

UNION Builds a new relation consisting of all tuples, and only those tuples, that appear in either or both of the parent (union-compatible) relations. (This is *not* the same as an **APPEND**, since the resulting relation cannot contain duplicate tuples.)

INTERSECT Builds a new relation consisting of all tuples, and only those tuples, that appear in both of the (union-compatible) parents.

DIFFERENCE Builds a new relation consisting of all tuples that appear in the first of the two (union-compatible) parent relations and not in the second.

JOIN (sometimes called an **INNER JOIN**) Builds a relation from two specified relations consisting of all possible concatenated pairs of tuples, one from each of the two parents, in such a way that in each pair, the two tuples satisfy some specified condition.

DIVIDE Takes two relations, one binary (i.e., having two attributes) and the other unary (having one attribute), and builds a relation consisting of all values of one attribute of the binary relation that match (in the other attribute) all values in the unary relation.

Since the result of using each of the above operations is again a relation, it is possible to nest operations at will. This property is known as *closure*. Note the similarity of this closure property with that of the normal algebra and arithmetic. An integer added to an integer gives an integer. In the same way, JOIN RELATION_1 to RELATION_2 gives a relation. The closure property in the relational algebra is important for exactly the same reason that it is important in the normal algebra and arithmetic.

SUP_ID	CITY	WAREHOUSE
S1	Detroit	30
S2	Boston	10
S3	Chicago	20
S4	Boston	10

Table 2

SUP_ID	PART_NUM
S1	P1
S1	P2
S1	P3
S2	P1
S2	P3
S3P4	
S3P5	
S4P4	
S4P5	
S4P6	

Table 3

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SUP_ID	CITY	WAREHOUSE
S1	Detroit	30
S2	Boston	10
S3	Chicago	20
S4	Boston	10

Table 4

decomposed, however, into two additional tables, which will incidentally bring the entire database into 3NF.

Third Normal Form

A relation is in 3NF if it is in 2NF and all nonkey attributes are mutually independent. Therefore, using the definitions above for CITY and WAREHOUSE, Table 2 is clearly not in 3NF.

This definition of 3NF, as originally defined by Codd, does not satisfactorily deal with relations in which the following conditions jointly apply:

- R has multiple candidate keys.
- The candidate keys are composite.
- The candidate keys share one or more common attributes.

Define a *determinant* as any attribute or collection of attributes on which some other attribute is fully functionally dependent. Now we can define a relation in Boyce/Codd Normal Form (BCNF) as one that is in 3NF and in which every determinant is a candidate key. (Many authors do not distinguish between 3NF and BCNF. I also shall not distinguish between them here except where explicitly stated.)

Without getting into the nasty details, we can attack the relation in Table 2 and, again through lossless projection into two tables, bring the entire database into 3NF. See tables 4 and 5.

The important thing to notice again is that joining these two tables over the attribute CITY exactly recreates the parent relation, but the update problem has been eliminated. So, we can easily get back to the original relation (the one that was only in 1NF).

SUP_ID	CITY	CITY	WAREHOUSE
S1	Detroit	Detroit	30
S2	Boston	Chicago	20
S3	Chicago	Boston	10
S4	Boston		

Table 5

Fourth Normal Form

Occasionally a situation will arise in which a relation, clearly in 3NF, will contain a large amount of redundant data. Many times these data will exhibit the following behavior:

If R contains the tuples (a, b_1, c_1) and (a, b_2, c_2), then R also contains (a, b_1, c_2) and (a, b_2, c_1).

For example, take the *unnormalized* relation shown in Figure 1. Product PR1 is manufactured in Detroit and Boston and consists of parts P1, P3, and P4; product PR2 is manufactured in Detroit and Chicago and consists of parts P1, P3, and P5. Dependencies of this form are known as *multivalued dependencies*.

Normalizing this relation creates a monster (see Table 6). This relation is already in 3NF, since the primary key is composite and consists of all three attributes. (If you don't understand why this automatically insures 3NF, take it on trust.) A projection can occur that will simplify the situation; however, it doesn't fit the old pattern (see tables 7 and 8). The database is now in 4NF and is required when multivalued dependencies are present.

Fifth (Projection-Join) Normal Form

Occasionally, relations will be encountered that cannot be decomposed into two projections without loss of information

PRODUCT	MFG PLANT	PART NUM
PR1	Detroit	P1
PR1	Detroit	P3
PR1	Detroit	P4
PR1	Boston	P1
PR1	Boston	P3
PR1	Boston	P4
PR2	Detroit	P1
PR2	Detroit	P3
PR2	Detroit	P5
PR2	Chicago	P1
PR2	Chicago	P3
PR2	Chicago	P5

Table 6

MULTI-USER
Get dBASE IV Source...
Code Base 4
The "C" Library for Database and Screen Management
WINDOWS
• Data Entry
• Menus
EXTENSIVE
• dBASE IV functionality
• Unlimited file opens
EFFICIENT
• Fast B+ tree indexing
• Small EXE files
COMPATIBLE
• Use dBASE or Clipper data and index files
Add the Speed and Flexibility of "C" to the Productivity of dBASE with a 30-day money-back guarantee. Only $295.
For a FREE DEMO Call Today! (403) 439-8171
SEQUITER SOFTWARE INC.
Number 269 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

but can be decomposed into three or more. Such relations are known as *n*-decomposable, meaning that the relation can be nonloss-decomposed into *n* projections but cannot be nonloss-decomposed into *m* projections for all $m < n$. Specifically, assume a relation R in which, for all *i* and *j* and attributes *A*, *B*, and *C*, a constraint of the following form holds:

If (A_i, B_i, C_j), (A_i, B_j, C_i), (A_j, B_i, C_i) appear in R,
then (A_i, B_i, C_i) appears in R.

Note in particular the cyclic nature of the constraint.

If any relation R exhibits a constraint of the above pattern, then R is said to be *n*-decomposable ($n > 2$). Note that, given the specific constraint described above, R can be decomposed into $R_1(A,B)$, $R_2(A,C)$, and $R_3(B,C)$. Also note that while, for example, $R' = R_1$ JOIN R_2 will in general produce spurious tuples, performing the additional JOIN with R_3 subject to the cyclic constraint will exactly reconstruct the original relation; that is, $R = R'$ JOIN R_3.

Specifically, take the following example:

ABC	A#	B#	C#
	--	--	--
	A_1	B_1	C_2
	A_1	B_2	C_1
	A_2	B_1	C_1
	A_1	B_1	C_1

decomposed by projection:

AB	A#	B#
	--	--
	A_1	B_1
	A_1	B_2
	A_2	B_1

BC	B#	C#
	--	--
	B_1	C_2
	B_2	C_1
	B_1	C_1

CA	C#	A#
	--	--
	C_2	A_1
	C_1	A_1
	C_1	A_2

then reconstituting by joining:

AB JOIN BC	A#	B#	C#	(joined over B#)
	--	--	--	
	A_1	B_1	C_2	
	A_1	B_1	C_1	
	A_1	B_2	C_1	
	A_2	B_1	C_2	(spurious)
	A_2	B_1	C_1	

(AB JOIN BC) JOIN AC	A#	B#	C	(joined over (A#,C#))
	--	--	--	
	A_1	B_1	C_2	
	A_1	B_1	C_1	
	A_1	B_2	C_1	
	A_2	B_1	C_1	

Such a decomposition, if the cyclic dependencies exist, places the relation in 5NF. Unfortunately, for reasons that I will get into later, one cannot be certain at a theoretical level whether or not a given relation is in 5NF, except for the simplest cases. Unsatisfactory as it is, we must be satisfied with "close enough for engineering use."

In a perhaps futile attempt to clarify an otherwise hopelessly confused concept, consider the situation of an auto parts distributor who uses two suppliers, Smith and Jones.

PRODUCT	MFG_PLANT
PR1	Detroit
PR1	Boston
PR2	Detroit
PR2	Chicago

Table 7

PRODUCT	PART_NUM
PR1	P1
PR1	P3
PR1	P4
PR2	P1
PR2	P3
PR2	P5

Table 8

The dealer purchases car doors and fenders from each, and each supplier provides these parts in both red and green.

Further, assume that Smith charges $75.00 for a door and $50.00 for a fender. He also attaches a 10-percent surcharge for red finish and a 12-percent surcharge for green. Similarly, Jones charges $80.00 for a door and $45.00 for a fender, with an 8-percent surcharge for red and 15 percent for green. All four combinations of parts and colors are purchased from each supplier. The relation describing this appears in Table 9. You can see that this relation is 3-decomposable (see Table 10).

PCM	P#	C#	M#	B	S
	Door	Red	Smith	75.00	0.10
	Door	Red	Jones	80.00	0.08
	Door	Green	Smith	75.00	0.12
	Door	Green	Jones	80.00	0.15
	Fender	Red	Smith	50.00	0.10
	Fender	Red	Jones	45.00	0.08
	Fender	Green	Smith	50.00	0.12
	Fender	Green	Jones	45.00	0.15

Table 9

Now, assume that Jones stops providing fenders. It is now necessary to delete only one record (from the PM relation) rather than two (from the PCM relation) in order to completely describe the new situation, and

```
(PC join PM) join CM
```

completely describes the new situation (see Table 11).

PC	P#	C#
	Door	Red
	Door	Green
	Fender	Red
	Fender	Green

PM	P#	M#	B
	Door	Smith	75.00
	Door	Jones	80.00
	Fender	Smith	50.00
	Fender	Jones	45.00

CM	C#	M#	S
	Red	Smith	.10
	Red	Jones	.08
	Green	Smith	.12
	Green	Jones	.15

Table 10

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PCM	P#	C#	M#	B	S
	Door	Red	Smith	75.00	.10
	Door	Red	Jones	80.00	.08
	Door	Green	Smith	75.00	.12
	Door	Green	Jones	80.00	.15
	Fender	Red	Smith	50.00	.10
	Fender	Green	Smith	50.00	.12

PCM = (PC join PM) join CM

Table 11

CM	C#	M#	S	
	Red	Smith	.10	
	Red	Jones	.08	(Deleted)
	Yellow	Jones	.11	(Added)
	Green	Smith	.12	
	Green	Jones	.15	

Table 12

Now consider the further situation in which Jones discontinues providing red finishes and introduces yellow. In the original case, the two records could be deleted from the PCM relation, but there would be no way to show the availability of yellow finish (unless we happened to have an immediate need for yellow parts). (From a strictly technical point of view, this is irrelevant, but operationally, the ability to preserve the information could be quite significant.)

It now becomes necessary only to make the obvious changes to the CM relation. See Table 12. Again, the correct relation PCM will result from performing the two-stage join.

Summary

In this article, I introduced the concept of normalization as a method for achieving data independence and eliminating data redundancy. I also introduced the First Normal Form (1NF), not because it is interesting in itself, but simply because we cannot get any further without it. I also gave examples of the types of problems attacked by 2NF–5NF without getting too far (I hope) into the underlying theory.

In subsequent articles, I will explore the second through fifth normal forms in all their glory. I will show by concrete examples how these can be used to decrease secondary storage usage, improve query performance, and simplify database maintenance.

See you soon!

Bill Madison is an independent computer consultant whose clients include Amtrak, the National Institutes of Health, the National Oceanic and Atmospheric Administration, the National Weather Service, the U.S. Department of Health and Human Services, and R. J. Reynolds Tobacco Co. Bill has been a member for nine years, and a chairman for seven years, of ANSI/X3/SPARC, the staff planning committee for the group within ANSI responsible for the development of industry standards relating to information systems. Readers may contact Bill at W. G. Madison and Associates, Ltd., P.O. Box 898, Greenbelt, MD 20770, 301-552-7234 (CompuServe 73240,342).

References

Codd, E. F., *A Relational Model of Data for Large Shared Data Banks*, CACM 13:6 (1970). (This paper is where it all started.)

Date, C. J., *An Introduction to Database Systems*, Addison-Wesley, 1986.

DBTG, *Report of the CODASYL Data Base Task Group*, April 1971, available from ACM, New York.

Tsichritzis, D.C. and Klug, A. (ed.), *The ANSI/X3/SPARC* "DBMS Framework: Report of the Study Group on Data Base Management Systems," *Information Systems* 3 (1978).

Wiederhold, G., *Database Design*, McGraw-Hill, 1977.

BRING DATABASE POWER TO YOUR SCREEN

Now you can achieve database flexibility in screen processing with the Radix Screen Management System. It's a language independent screen and keyboard controller. The package includes a facility that allows full screen editing of screen maps. Once created, a screen map is allocated and released at runtime, no compiling or linking necessary. Use the screen map in different applications developed under different languages. There are only eight commands to learn which control the screen maps. In addition, an Internal Document Facility will automatically display and scroll a window of your application's help text.

CALL TODAY FOR INFORMATION
201-697-2630

Other features include:

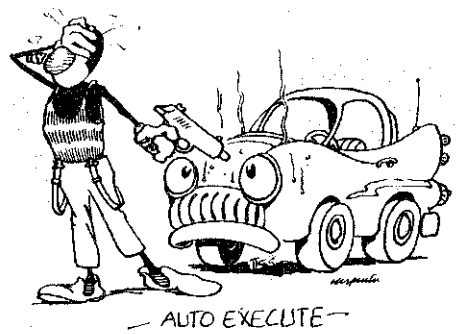
- **Window and Overlay Screens**
- **Alter Attributes at Runtime**
- **Move Data Field-by-Field or in groups**
- **Request Only Modified Values**
- **Update the Screen Map Anytime**
- **Extensive Documentation**
- **Uses Standard Command Interface**

To order, send $198 plus $7 for shipping. No royalties.

Requires an IBM PC or compatible, 512KB of memory, and DOS 2.0 or higher. UNIX and OS/2 versions will be available soon. For more information call (201) 697-2630, or send $5 for demonstration diskette and Concepts & Facilities manual.

RADIX SYSTEMS INC.
P.O. Box 306
Rochelle Park, NJ 07662

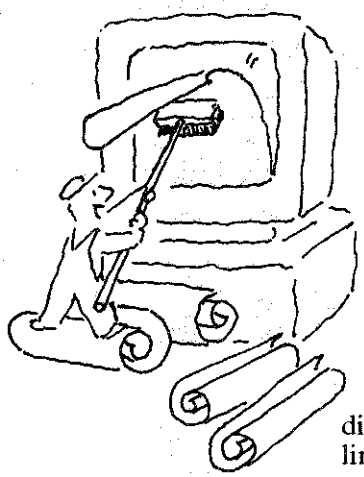
Number **251** on the Reader Service Card



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Two great tools.

SAYWHAT?!
The lightning-fast screen generator.



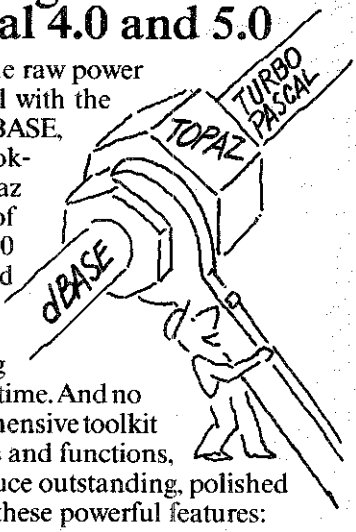
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ
The breakthrough toolkit for Turbo Pascal 4.0 and 5.0



If you'd like to combine the raw power and speed of Turbo Pascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz (our brand new collection of units for Turbo Pascal 4.0 and 5.0) was specially created to let you enjoy the best of *both* worlds. The result? You can create truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 200 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like **SAY, GET, PICTURE, RANGE,** color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 7 indexes each with database routines like **USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO,** and **FIND.**
- No need to buy dBASE. **CREATE, BROWSE** and **REPORT** utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- **BROWSE** any DBF file with just one line of code! Programmable and windowed too.
- Comprehensive Time & Date math in 7 international formats.
- Powerful *code* and *REPORT generators* included!
- Comes with a complete 250 page manual, plus sample programs to get you started.

$49.95
Special introductory price! (Until March 31, -then $74.95)

ORDER NOW.
YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $15. Buy Saywhat?! and Topaz together before March 31, 1989 for just $85 (plus $5 shipping & handling).

Visit your nearest dealer or call toll-free:

800-468-9273

In California: 800-231-7849
International: 415-571-5019

The Research Group
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany (2534-7093)

Guaranteed!

THE RESEARCH GROUP

Number 198 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Future Arrives

by
Tom Swan

© 1989 by Tom Swan

The Federal Express driver didn't know it, but last week he handed me the future of Pascal programming. Actually, I didn't know it either—didn't know what a jewel lay inside that red-and-white packet, not even after I had zipped it open. But then, after spending several days with Turbo Pascal 5.5, I began to get the picture, which I'll try to draw for you here with words. In my opinion, Borland's newest release of Turbo Pascal is the most exciting development in Pascal programming to hit the streets in a decade. Don't miss this upgrade!

What excites me the most are the new object-oriented programming (OOP) extensions that will take Turbo Pascal developers kicking in high style into the 1990s. Those of you who are familiar with OOP understand the importance of these techniques for solving ever more complex programming problems. Those who haven't yet caught the OOP fever now have an alternative to C++ in Turbo Pascal 5.5, which retains all of the speed and utility that has made Borland's best-selling compiler famous. If you know a lot about Turbo Pascal but little or nothing about OOP, trust me. The only discovery of equal importance I've stumbled upon in recent years is jalapeo jelly and cream cheese on a rye bagel. But that's only food for thought. Turbo Pascal OOP is "food that thinks." More on this later.

Objects aren't all that's hot in Turbo Pascal 5.5. The complete list of additions and improvements over the previous 5.0 release includes:

- Object-oriented-programming extensions
- Improved overlay manager
- New Turbo Debugger 1.5
- Ten-percent improvement in compilation speed
- Enhanced on-line help and a disk-based tutorial

I'll cover the new OOP extensions and improved overlay manager in a moment.

The new Turbo Debugger 1.5 is sold either separately or as part of the Turbo Pascal Professional package. A stripped-down version of the debugger is included in the Turbo Pascal 5.5 integrated development environment. The new debuggers recognize Turbo Pascal's object-oriented extensions but otherwise are unchanged. The claim of a ten-percent compilation-speed increase appears to be accurate, although I didn't put the compiler through any rigorous benchmarks. A ten-percent improvement isn't all that thrilling, of course, but it *is* remarkable when you consider that Turbo Pascal is already the fastest compiler alive and it now has all these additional object-oriented constructions to round up.

I asked Borland's programmers how they managed to improve performance while adding new features; they told me that whenever they want to shave compilation times, they just remove a few hidden wait loops that were built in to ensure good reviews and benchmarks. Right. Remember, you heard it here first.

Beginners will appreciate the enhanced on-line help, which now allows you to cut sample code from help screens and paste the text into programs. (This feature didn't work very well in the beta-test software I have, but maybe improvements will be made to the final release.) The promised disk-based tutor wasn't included with my review copy.

Tweaking Overlays

Turbo Pascal 5.5's improved Overlay unit reduces disk thrashing by employing a clever trick that puts overlaid code "on probation" before releasing it from its RAM-buffer prison. This helps to keep frequently used overlays in memory, thus reducing disk accesses and improving performance. The new overlay management scheme is completely compatible with 5.0 source code.

The Overlay unit has several new variables, a new function pointer data type, a new procedure, and a new function. You can use these features to fine-tune overlay performance. You also can attach overlays to a compiled EXE file—you don't have to store overlays in separate OVR files anymore. The new unit declarations are:

```
TYPE  OvrReadFunc = FUNCTION( ovrSeg : Word)
                    : Integer;

VAR   ovrReadBuf    : OvrReadFunc;
      ovrTrapCount  : Word;
      ovrLoadCount  : Word;
      ovrFileMode   : Byte;

PROCEDURE OvrSetRetry( size : LongInt );
FUNCTION OvrGetRetry : LongInt;
```

(Note: Borland capitalizes variables; I don't. I type keywords in all uppercase; they don't. For consistency with other example code in this article, I've modified the Overlay unit declarations to match my style. These changes are purely cosmetic.)

The **OvrReadFunc FUNCTION** data type and variable **OvrReadBuf** let you hang your own code onto the routines

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

that load overlays into memory. The method for accomplishing this resembles the way you can chain custom exit procedures together for executing cleanup code just before a program ends. Similarly, you can attach custom code to the overlay manager, perhaps adding a prompt to insert a diskette, improving error recovery, executing debugging statements, or performing other operations before or after attempting to load an overlay into memory.

New variables **ovrTrapCount** and **ovrLoadCount** keep running statistics on the number of times overlays are requested and loaded. Monitoring these values with the integrated or stand-alone Turbo Debuggers will help you to tweak overlay performance. My advance copy of the Turbo Pascal update guide is somewhat unclear on how to use these values, but I suspect that the goal is to achieve a high **ovrTrapCount** (indicating a "hit" on an overlay currently on probation but still in memory) versus a low **ovrLoadCount** (indicating how many times overlays are loaded from disk).

Another variable, **ovrFileMode**, lets you pass an access code to the DOS routines responsible for loading overlay code from disk. The purpose of **ovrFileMode**, normally set to 0 (read-only access), is to allow shared access to overlaid programs over a network. (I did not test this feature, so I'm merely quoting the advance documentation here.)

The new Overlay unit also adds one procedure and function. **PROCEDURE OvrSetRetry** adjusts the size of the "probation area" in the overlay buffer. **FUNCTION OvrGetRetry** returns the current setting of this value. Normally, the probation-area size is 0, which makes the new Overlay unit operationally compatible with 5.0 but it also disables "probation-reprieval" memory management. Borland recommends changing the probationary-buffer-size value to about 1/3 or 1/2 of the total overlay buffer size, but you'll have to experiment to find the optimal value for each program. You can then set the probation-buffer size, which also enables the new memory-management code, with statements such as

```
OvrInit( 'OVERUNIT.OVR' );
OvrSetBuf( BufferSize );
OvrSetRetry( BufferSize DIV 3 );
```

In addition to these new Overlay unit features, you now can attach compiled OVR files to the end of a compiled EXE file. (You can't do this to EXE files containing Turbo Debugger information, though. For debugging overlaid programs, you must store overlays in separate files.) Incredibly, you use the DOS **COPY** command's */b* (binary) option to join programs and overlays. For example, to attach OVERUNIT.OVR to MAIN.EXE, you simply type:

```
copy /b main.exe + overunit.ovr
```

You also have to change the **OvrInit** call to **OvrInit(ParamStr(0))**, telling the overlay manager that the overlaid code is now at the end of the EXE file.

Programming with Objects

There isn't enough room here to describe Turbo Pascal 5.5's new object-oriented extensions fully. Turbo Pascal OOP is a story that deserves a patient telling, and in the coming months I'll be relating my experiences with OOP techniques in a series of *PJ* articles. For now, I'll limit the discussion to a few general observations.

The first rule is to remember that Turbo Pascal OOP extensions are optional. The upgraded compiler is 100-percent compatible with version 5.0 syntax. This means that beginners can learn Pascal fundamentals and postpone OOP business until later (although there's no reason beginners can't learn OOP techniques). Experts can continue working for a living while selectively adding OOP tools to programs without having to delay development or learn a new language.

What Is an OBJECT?

An **OBJECT** is a new data type, resembling a **RECORD** type but with several new capabilities.

Earlier, I said that Turbo Pascal OOP is more akin to "food that thinks" than "food for thought." I'm not just being flip. Common Pascal procedural programming requires you to think in terms of processes that operate on data—that is, the food for thought. In OOP, objects contain not only data but also actions, called *methods*. Food doesn't think, of course, but locked inside food is the energy that causes thought (as well as other action). You are what you eat, and in this way, OBJECTs are like food that thinks—data with the energy to cause action.

In Turbo Pascal, **OBJECT**s resemble **RECORD**s. Inside an **OBJECT** are fields, which may be of different data types. In addition to data fields, an **OBJECT** also declares the methods that the object knows how to perform. For example, here's a simple **OBJECT** named **Method**:

```
Message = OBJECT
    theText : String;
    PROCEDURE Init( s : String );
    PROCEDURE Show;
END;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

In this **OBJECT**, data field **theText** stores a string of text. Most **OBJECT**s have one or more such fields. There are also two methods, **Init** and **Show**, declared as **PROCEDURE**s. (Methods may also be **FUNCTIONS**.) Elsewhere, the program would supply bodies (not shown here) for each declared method in all objects.

A key OOP concept is that **OBJECTS** can inherit the characteristics of other objects. Inheritance lets you build complex descendant objects from ancestor objects. For example, another object can inherit the characteristics of **Message** by including the ancestor object name in parentheses after the **OBJECT** keyword:

```
NewMessage = OBJECT( Message )
    PROCEDURE Show;
END;
```

NewMessage (the descendant **OBJECT**) acquires all the characteristics—that is, the data and method fields—of **Message** (the ancestor object). At the same time, **NewMessage** replaces one of those methods with its own **Show** method. The new object might add other data fields and methods, too. In this way, descendant objects inherit and extend the capabilities of their ancestors. Extensibility is another key OOP concept.

Interestingly, the source code to the original ancestor **OBJECT** is not required in order to extend that **OBJECT**. If you purchase a compiled toolkit, even if the source code is not available, you can extend **OBJECT** definitions, and all existing programming will work with your new additions. A typical example is a graphics library with various shapes that know how to rotate, stretch, reposition, fill, and perform other actions on themselves. With OOP, you can add additional **OBJECT**s to this library, and the existing code will work as it did before. Imagine having a CADD system that allows you to program custom **OBJECT**s without changing a single line of existing source code. Providing this same degree of extensibility with procedural language tools is difficult, if not practically impossible.

Now, I have to stop this brief look at OOP, because accomplishing such lofty goals requires a full understanding of something called *Virtual methods*. The previous simple examples declared only *Static methods* and only begin to scratch the surface of OOP possibilities. Choosing whether to make a method Static or Virtual is one of OOP's darker alleys, one that we'll have to visit in the light of another day.

Other Turbo Pascal OOP extensions provide ways to construct dynamic **OBJECT**s that know how to create and destroy themselves in memory by way of **CONSTRUCTOR**s and **DESTRUCTOR**s—two more keywords newly added to Turbo Pascal. Extended forms of **New** and **Dispose** are provided for more easily using these advanced techniques. C++ programmers will recognize much of this—Turbo Pascal OOP is patterned after C++ concepts.

I'm tempted to continue, but I want to start off on the right foot, presenting good examples of useful code, both to explain OOP techniques and to share practical tools that you can incorporate into your own projects. This is not the time or place for a tutorial. Until next time, then, I urge you to place your order for Turbo Pascal 5.5. The time for the future is now! PJ

Contributing editor Tom Swan is the author of Mastering Turbo Pascal 4.0 *(1988) and* Mastering Turbo Assembler *(1989), published by Howard W. Sams.*

TechnoJock's Turbo Toolkit

Attention Turbo Pascal programmers.....

Cut your program development time in half and create sophisticated user interfaces in minutes with *TechnoJock's Turbo Toolkit!*

The *Toolkit* contains twelve units with all source code included, and a 300 page manual full of examples. These range from a comprehensive unit that provides a Sidekick Plus style directory window with all the expected sorting features, to a basic procedure that writes text centered on the screen.

And it's easy to use! If you've bought other toolkits but didn't use them because of all their complicated rules ... TTT is for you.

TECHNOJOCK SOFTWARE

- **ultra fast screen writing**
- **EGA/VGA extended lines**
- **exploding boxes**
- **virtual screens**
- **sorted directory listings**
- **comprehensive form input**
- **pulldown menus**
- **nested menus**
- **string manipulation**
- **windows**
- **full mouse support**
- **keyboard control**
- **date manipulation**

"TechnoJock's Turbo Toolkit may just cut your work in half! It provides easy-to-implement procedures.... designed for all Turbo Pascal programmers. With this Toolkit everyone becomes a TechnoJock."
Turbo Users Group

"Here, despite the whimsical name, is shareware at its best: an innovative product, a superb manual, quick mail-order service, and a reasonable price... If you're programming in Turbo Pascal, you absolutely must have TTT!"
Public-Domain Software & Shareware 2nd edition

"Looks great! We have decided to implement your Toolkit in in-house programs for Tech Support."
Rich Kubota
Technical Support
Borland International

TechnoJock Software, Inc.
PO Box 820927
Houston, TX 77282-0927

Orders: 1-800-242-4775
Enquiries: (713) 493-6354

Only $49.95!

Plus $5.00 shipping and handling (Canada $10.00, overseas $15.00). Texas residents add 8% Sales Tax.

Send checks to TechnoJock Software. Send Visa/Mastercard details to:
Public (Software) Library
PO Box 35705
Houston TX 77232-5705

Number **289** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Lint Utilities

by

Rex Jaeschke

© Rex Jaeschke. 1989. All rights reserved.

This month we'll take a break from our ANSI review of compilers to take a look at the UNIX C programmer's favorite companion, Lint. Lint is a static analysis tool—it analyzes source code rather than tracking the program at execution. As such, Lint does not measure performance, memory and disk usage, and the like, since none of these aspects can be determined without actually following one particular set of logic paths.

There are a multitude of tools available to help you develop quality C programs in an economical manner, and while some of them overlap slightly in function, for the most part they each have different primary goals. Examples of these tools include syntax-directed editors, compilers, and interpreters, symbolic debuggers, profilers, source code control systems, system build utilities and run-time libraries.

Since many—most?—programmers spend an inordinate amount of time in the debugging phase, it would seem appropriate to have a tool to help with that phase. Traditionally, at least in non-UNIX worlds, the tools available were the language translator and, if you were lucky, a symbolic debugger.

Also traditionally, a compiler's job was to diagnose syntax errors and constraint violations and, in the absence of these, to generate object code. Modern-day UNIX compilers (and other tools) still follow this philosophy. UNIX tools typically do one thing and do it well. For example, if you want a compilation listing, don't expect the compiler to produce one; use another utility (perhaps written by you) to do that. Compilers are for compiling, right?

The problem with this narrow approach is that eliminating the errors detected by the compiler is almost always a trivial task. When your source compiles without error but does not execute as expected, the real debugging effort is expended. We're talking about logic errors and errors introduced through carelessness or lack of understanding about a particular operator, construct, or library routine. Of course, symbolic debuggers can help discover such problems, but at a significant cost—you almost always have to do it interactively, and you have to know where to start the search. (In fact, the approach I teach regarding the use of debuggers is that you have no business starting one up unless you have a pretty good idea of what you are looking for and the section of code you are interested in. You can't afford to single-step through every instruction every time.)

As this column is about Lint, you might have guessed that it is the missing link, and to a significant extent it is—or at least it *can* be, depending on how your Lint is implemented).

To put Lint in perspective, let's go back to the reason for its birth. Sometime in the mid 1970s, Stephen Johnson, a mathematician at Bell Labs, started to write the first version of the portable C compiler (pcc). This compiler became the basis on which all production UNIX C compilers were (and, I believe, still are) based.

While developing this compiler, Johnson decided he needed some sort of tool to help him debug it, and it occurred to him that he could build one without too much difficulty. So, he took the front end of the compiler he was building and instead of having it generate object, he added a considerable amount of extra checking of the source and had it produce a "quality of implementation" report. For example, it reported on assignments of values whose type has more precision or range than the destination type. It identified unreachable code, and, most important, it could look at separate source modules at the same time to check function call linkage and report on argument number and type mismatches. (There were quite a few other checks done, as well.)

As to the name Lint, when I interviewed Johnson several years ago, he said that the tool didn't initially have a name. But one day, as he was cleaning the lint filter of his clothes dryer, it occurred to him that that was what his program was doing: getting the fluff out of other programs. Cute, but true.

Warning: To quite a few people, Lint is a religion, so much so that they won't even attempt to compile their code until it has gotten through Lint without error. And, depending on the version of Lint you use, you may have to jump through some hoops to "shut it up." It sure can produce a large number of messages, many of which you might want to ignore. The trick though, is to switch off those you don't care about while keeping all those you do. It takes some practice to decide how to approach this.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Current Status of Lint

Back in the good old days, C was pretty much restricted to UNIX environments, and since Lint was bundled with UNIX, there was no market for a separate product. However, once C became very popular outside UNIX—primarily when DOS C systems began flooding the market in 1982 and 1983—DOS C programmers who knew about Lint had to go without it, since no comparable tool was available under DOS.

Some time later, two independent DOS versions of Lint emerged, one called Pre-C, from Wizard Systems Software in Arlington, Mass. (and marketed by Phoenix Computer Products), the other called PC-Lint, from Gimpel Software in Collegeville, Pa. (Gimpel also sells a version for the Amiga and a generic version you can port to other systems, such as VAX/VMS.)

Pre-C, written by Bob Jervis of Wizard, was withdrawn from the marketplace in late 1988. I presume that when Bob and his compiler sold out to Borland (re-emerging as Turbo C), Phoenix was no longer able to update the source for Pre-C. And, considering the many changes that occurred to the standard C environment in the last few years, Pre-C was no longer compatible with modern compilers.

Note though, that while Jervis created Pre-C, he also incorporated many of its capabilities into his compiler, Wizard C. And since Turbo C is derived directly from this compiler, Turbo C has a good many Lint capabilities available through its interactive options command. For this reason, Turbo C is my premier interactive C development environment.

PC-Lint, on the other hand, continues to thrive, with version 3 being shipped in late 1988. And each version gets more comprehensive.

A relative newcomer has also entered the Lint stakes: MMC AD Systems and their C Programmer's Toolbox Volume II contains a tool called CLint. We will take a look at both PC-Lint and CLint in this column.

Of course, Lint continues to be shipped with UNIX systems, and you can expect that with new releases, Lint will be updated to reflect the C compiler it's shipped with. Surprisingly, Lint does not seem to have been made available in many non-DOS, non-UNIX environments. Apart from a VAX/VMS version I've heard of and Gimpel's Amiga and generic Lints, you're out of luck. (MMC AD also has McCLint for the Macintosh.)

You should be aware that all Lints are not created equal. All third-party versions are written from scratch and were originally intended to mimic Lint under UNIX. Some fall short, others are comparable, while others vastly exceed UNIX Lint's capabilities. Specifically then, there is no such thing as a product called Lint; there are simply various implementations of the original UNIX Lint idea.

PC-Lint V3.00

The 125-single-sided-page manual seems somewhat imposing. However, if you want to take maximum advantage of this implementation's many and varied capabilities, you'll find them all necessary. Using PC-Lint is not simply a matter of saying, "Check my code." Depending on the origins of your sources and their vintages, you might need to do a reasonable amount of checking and experimenting before you settle on an appropriate set of processing options.

The following small example should give you a feel for the kinds of things PC-Lint can check for:

```
#include <string.h>

#define SIZE1 4
#define SIZE2 4U
void f(unsigned int);
void g(void);

main()
{
    int i = -4;

/*13*/      f(10);
/*14*       f((unsigned int)10);
/*15*/      f(10u);

/*17*/      if (strlen("abcdef") > 7)
            ;

/*20*/      if (strlen("abcdef") > -7)
            g();

/*23*/      if (strlen("abcdef") > i)
            g();

/*26*/      if (sizeof("abcdef") > 7)
            g();

/*29*/      strncmp("abcdef", "abcdexyz", 4);
/*30*/      i = strncmp("abcdef", "abcdexyz", SIZE1);
/*31*/      i = strncmp("abcdef", "abcdexyz", SIZE2);
/*32*/      i = strncmp("abcdef", "abcdexyz", i);

/*34*/      printf("%d %10.4f\n", 1.23);

/*36*/      if (i = -3)
            g();
}
```

It was run with only one message type being disabled. Of course, the source file is contrived to be suspect, but that's exactly how such a tool is checked. Note, though, that the file is ANSI conforming; an ANSI-conforming compiler should produce no errors. Also, it is not required to produce any warning or informational messages. The output produced by the file included the following:

```
---- Module c:\c\pc-lint\tbc.c

---- Module test.c
File test.c, Line 20
    if (strlen("abcdef") > -7)
    ^
    Warning 548: else expected

File test.c, Line 20
                              ^
    Warning 574: Signed-unsigned mix with relational
File test.c, Line 20
    Info 737: Loss of sign in promotion from int to
        unsigned int
```

PC-Lint can be configured to work with code intended for almost any compiler (even those running outside of MS-DOS.) However, you need to tell it which headers to use; I chose Turbo C V2.0. (I also checked it out with WATCOM V7 and Microsoft V5.1.)

The first message takes a little interpreting, although the manual's explanation is quite clear: "A construct of the form if(e); was found which was not followed by an else. This is almost certainly an unwanted semicolon as it inhibits the if from having any effect." Quite true, and the statement is reasonably suspicious. Note though, that the message really applies to the statement beginning on line 17.

When I ran PC-Lint over quite a number of my own source files, I discovered that I consistently mixed signed and

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug Multi-Megabyte DOS Programs

There was a time when the only way to debug very large DOS C programs meant suffering under 640KB memory constraints. Not any longer. *Instant-C* opens a new age of debugging under DOS.

Instant-C is specifically designed to debug large DOS programs. It works in "protected mode" on all 80286 and 80386-based PCs giving you access to **16MB** of extended memory. *Instant-C*'s source-level debugging lets you debug and run programs in excess of 100,000 lines of code without worrying about DOS memory limitations or resorting to overlays. Hard to believe? It's true. No other DOS debugger can live up to *Instant-C*'s standards.

And *Instant-C*'s debugging capabilities are unsurpassed. With features such as automatic run-time checking, data monitoring, source code animation, source code backtracing, and unlimited conditional breakpoints at your disposal, debugging your large programs will be faster than ever before.

Instant-C also includes an integrated C compiler, linker, editor, and source formatter. With *Instant-C* you get a powerful integrated tool package for working with your C programs. And *Instant-C* is compatible with your Microsoft C compiler.

Save yourself hours of debugging time every single day. Try *Instant-C* and discover how to make debugging your large DOS C programs quicker than you ever believed possible. With *Instant-C*'s **30 day no risk money back guarantee** the only thing you've got to lose is bugs.

- **Debug Multi-Megabyte Programs**
- **Automatic Run-time Checking**
- **All Debugging in C**
- **Unlimited Conditional Breakpoints**
- **Compatible with your Assembly Code**
- **Source Code Backtracing**
- **Support for Full C Language**
- **Data Breakpoints**
- **Source Code Animation**
- **Compatible with Microsoft C**

Rational Systems' 30 Day No Risk Money Back Guarantee

If for any reason you decide not to keep *Instant-C*, just return it to us within 30 days of purchase and we will gladly refund your money.

Order *Instant-C* today!

***only* $795**

Rational
Systems, Inc.

220 North Main Street
Natick, MA 01760
Tel: **(508) 653-6006**
FAX: **(508) 655-2753**

Trademarks are acknowledged to Rational Systems, Inc., and Microsoft Corp.

Number **20** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

unsigned operands with relational and equality operators. And while I did not find any such uses erroneous, I did vow to conscientiously try to avoid them, since most of the code I write has to be quite portable. Certainly it's illogical to use a signed type when no negative values are possible. Also, if you compare a negative value with an unsigned, you can get surprising results. The problem has escalated in recent years since ANSI C (and most modern compilers) have *strlen* and *sizeof* giving an unsigned type, rather than the traditional signed type.

```
File test.c, Line 23
    if (strlen("abcdef") > i)
                         ^
    Warning 574: Signed-unsigned mix with relational

File test.c, Line 23
    Info 737: Loss of sign in promotion from int to
        unsigned int

File test.c, Line 26
    if (sizeof("abcdef") > 7)
                         ^
    Warning 506: Constant value Boolean
```

Since the outcome of the controlling expression in line 26 is a constant whose value is known at compile time, code for the if test might not even be generated. However, the operator to *sizeof* might not be so simple as this; for example, it might be the name of an array whose dimension is conditionally compiled. Still, most occurrences of this message can safely be ignored; so, you have to decide among disabling this message globally on the command line and getting no such messages at all, keeping it and wading through and ignoring those you can, or switching it on and off in the source code as you need it.

```
File test.c, Line 29
    strncmp("abcdef", "abcdexyz", 4);
                                 ^
    Warning 534: Return mode of strncmp inconsistent
        with line 40, file
    STRING.H, module c:\c\pc-lint\tbc.c
```

Here, we have called a nonvoid function and ignored its return value. This is common with several of the **str*** functions and also with the **printf** family. Since the return value is for convenience only, it is perfectly all right to ignore it. For each compiler you use, you need to have a profile for use by PC-Lint. This source file contains PC-Lint commands that define certain characteristics of your compiler and runtime library. For example, you can identify all functions whose return value can safely be ignored so that PC-Lint will not complain about such usage.

```
File test.c, Line 32
    i = strncmp("abcdef", "abcdexyz", i);
                                      ^
    Info 732: Loss of sign (arg. no. 3) (int to
        unsigned int)
```

The prototype for strncmp declared the third argument to have type *size_t*, an unsigned integer type, and while i's type int is assignment compatible with *size_t* as required by ANSI C, clearly, passing a negative value could result in unexpected behavior.

```
File test.c, Line 34
    printf("%d %10.4f\n", 1.23);
    ^
Info 718: printf undeclared this module, assumed to
        return int

File test.c, Line 34
                                  ^
    Warning 558: number of arguments inconsistent
        with format

File test.c, Line 34
    Warning 559: size of argument no. 2 inconsistent
        with format
```

Since stdio.h is not included and **printf** is not explicitly declared, **printf** is flagged as an unknown function. (Any tool that identifies a function being called not in the scope of a function prototype will likely save your bacon on many occasions. Forgetting to include a header is a very common source of error.)

A welcome invention of Gimpel's is a check of the edit masks used in calls to the **printf** family. In this case, there are two masks and only one argument, and that is not compatible with the first mask. PC-Lint also flags ill-formed and unrecognized edit masks. (It also can handle **scanf** calls and make sure arguments are passed by address.)

```
File test.c, Line 36
    if (i = -3)
           ^
    Info 720: Boolean test of assignment

File test.c, Line 36
    Warning 506: Constant value Boolean
```

This case occurs quite often, particular when programmers are new to C or they switch between languages on a regular basis. It almost always is an error, in that you want == instead. PC-Lint reports on the use of assignments in contexts where a truth value is required to flag just this problem. If you really want to have such assignments, you can use

```
((i = -3) != 0)
```

instead. It should generate the same code, and it explicitly declares your intent to some unsuspecting maintenance programmer at a later date.

```
File test.c, Line 38
}
^
    Warning 533: Return mode of main inconsistent
        with line 10
```

Function **main** has a return type of int, and since there is no return statement with an int value, this message is produced. Gimpel suggests you define **main** to be a void function in the profile file, and assuming you never explicitly return from **main**, that's not unreasonable.

```
Warning 526: f (line 6, file test.c) not defined
Warning 526: g (line 7, file test.c) not defined
```

Commands may be given to PC-Lint in one of two (and sometimes both) ways: on the command line or in the source file. When used in the source, they have the following general form:

```
/*lint option1 option2 ... [comments] */
```

very much like UNIX Lint. As such, these "special" comments can be placed throughout your source and only be

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

recognized by PC-Lint. (ANSI C has generalized this approach via the addition of **#pragma**.)

Messages are arranged in hierarchies of tens and hundreds and can be enabled or disabled individually or by tens or hundreds groups. For example, messages 700–799 are information messages. Commands of -e732, -e73?, and -e7?? disable the corresponding message or groups of messages, respectively; a "+" is used to enable a message. Using this approach, you can directly control the granularity of message reporting for syntax errors, internal Lint errors, fatal errors, warnings, and informational messages. Of course, using more than a few such commands can quickly exceed your command-line buffer size; therefore, you use the profile approach. And for each project, you can have a different profile. In fact, you can have a core standard profile and include it in each project's customized profile.

Options exist to define the size of any type so you can process code intended for various memory models under DOS or even for non-DOS architectures. Optionally, dollar signs are permitted in identifiers, and extended keywords can be defined, making it relatively easy to customize PC-Lint to process VAX C source, for example.

The usual options provided with modern compilers are also available—for example, whether a plain CHAR is signed, the type of *size_t* and *ptrdiff_t*, nested comment support, defining macros, specifying header paths, preprocess only, and use ANSI C mode. You have considerable control over the format and brevity of messages, and this is most welcome, since you typically have to wade through a lot of output in the early stages until you either "correct" your programming habits or disable various message categories.

PC-Lint goes well beyond the original Lint model. For example, it detects "gray" expressions, where it appears the programmer relied on undefined order of evaluation, as in "i++ + i". It inspects **printf** and **scanf** family edit masks and argument lists, complains if you initialize static objects to zero by default, and even skirts the religious war of brace positioning. It does this by complaining if you seem to have less indentation than established by your brace positions. If you don't like this, you disable it—it's that simple.

The manual states, "PC-Lint assumes the ANSI C definition of C and supports K&R where it does not conflict with ANSI. It also supports common extensions to the standard especially for specific compilers." I have found this to be very accurate, with only a few rough edges regarding ANSI C conformance. Apart from performing a complete syntactic check, PC-Lint has more than 130 "quality of implementation" checks, most of which are not done by even the most modern compilers. And the list of checks increases with each new release.

For much of its life, PC-Lint has been the only game in town. In such circumstances, many vendors have the attitude "Why improve the product if we don't have to? There's no real competition." This has not been the case with Gimpel. Each release has gotten progressively better. At $139, the product is a steal. Once you have set up the product for a given project, I'd be very surprised if it didn't pay for itself in the first half-day of serious use.

CLint V1.1

When I first read the CLint manual, I was very disappointed, since it stated that function prototypes were not supported. I couldn't imagine how such a tool would be useful in this age of modern C. As it happens, the latest release, V1.1, not

PC-lint *The # 1 Lint for MS-DOS*

```
          void
        main() (
    struct{unsigned
   a:2, b:3, c:4;}s;
 unsigned row,col;char
ch;int x;char y;unsigned
z;void          printf ( )
; /*             a lie but
                  harmless
                  */ x=0 ;
                   /* x is
                    set  to
                   zero*/row
                   =25;col=
                  80; if (
                 row+col>
               -1)x++;ch=
             '\xFF';if(
           (unsigned)
         ch==0xFF)
       x++;z=2;
       y=(z &
       0x3)<<
       7; if(
       y)x++;
       z =12;


         if(z/
      -1)x++;s.b
      =10;if( s.b
      >5) x++;z =
      37;if(-z<0)
      x++;printf(
       "x=%d\n",
         x );}
```

You have just inherited this C program.

What do you do?

(a) Run away

(b) Run PC-lint

Use PC-lint to examine foreign looking C code and report on potential problems and difficulties. PC-lint's many options let you quickly and easily suppress unwanted error messages and extract just the information you need.

PC-lint is a C source code analysis tool. It will check code more thoroughly than a C compiler and, because it looks across multiple modules, it enjoys a perspective that your compiler doesn't have.

Control your development time and cost. Locate source code errors that could otherwise go undetected for days, months or even years.

Over **200 error messages**. More than **70 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, both, etc. **Easy to Use** - If you know C you already know how to use PC-lint. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI **prototypes** for your K&R functions.

Mainframe and Mini Programmers

Get **FlexeLint** (formerly Generic Lint) in shrouded source form for VAX/VMS, Unix, DG AOS/VS, OS-9, Apple MPW, IBM VM/MVS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.

Gimpel Software

3207 Hogarth Lane Collegeville PA 19426

(215)584-4261

Only $139 - CALL TODAY

30 Day Money-back Guarantee

Specify MS-DOS or OS/2. Quantity and educational discounts available. PA add 6% sales tax -- Outside USA add $20.
PC-lint and FlexeLint are trademarks of Gimpel Software.

Contact this advertiser directly

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

only supports them but also supports a great deal of the proposed ANSI C standard. The problem is that the manual set was not updated for V1.1; you have to read the release notes on disk.

CLint is one of many C development tools packaged as a two-volume set called "The C Programmer's Toolbox." (The tools include versions of some popular UNIX commands, as well as other useful things.) With CLint, as with most toolboxes, you get a wide variety of goodies, none of which is particularly stellar.

To be sure, V1.1 is a credible implementation of Lint and, together with a bunch of utilities, it sells for $80. (Volume I also costs $80.) If you can use the other tools, CLint is a nice bonus. However, given the enormous capability of PC-Lint, CLint has some stiff competition.

To become a bigger force in the Lint arena, CLint needs more than 20 pages of documentation. There is no detailed list or explanation of the error messages, and so it is not always clear just what was being tested in a given situation. CLint has very few options available in comparison with PC-Lint. You can chose between only the small- and large-memory models, and you cannot establish the usual defaults that most modern compilers provide (for example, whether or not plain CHARs are signed).

The manual states, ". . . CLint expects successfully compiled source code. CLint does not have full error recovery facilities. . . ." Although this is contrary to the traditional idea of Lint, it isn't really a drawback, since your compiler can do that checking. But it's nice to get a second opinion confirming that compilers can and do fail to detect errors.

Except for a couple of changes to the test program shown earlier (CLint didn't recognize unsigned constants), CLint processed that file, producing the following output:

```
====================
test1.c:23  if (strlen("abcdef") > i)
                                  ^
*** Warning ***  12-Assignment or comparison type
       mismatch
--------------------
test1.c:31  i = strncmp("abcdef", "abcdexyz", SIZE2);
                                                  ^
*** Warning ***  9-Type mismatch
--------------------
test1.c:32  i = strncmp("abcdef", "abcdexyz", i);
                                               ^
*** Warning ***  9-Type mismatch
--------------------
test1.c:34  printf("%d %10.4f\n", 1.23);
            ^
*** Warning ***  5-Undeclared function (printf)
*** Warning ***  35-Expected function (printf)
       argument types unknown
--------------------
test1.c:36  if (i = -3)
                  ^
*** Warning ***  14-Non-logical top condition
       operator

            *** Total Processing Summary ***
Total syntax errors:   0  # files checked:       3
Total syntax warnings: 6  Total lines checked: 140
                          # files scanned:       3
                          Total lines scanned: 140
```

CLint is a nice bonus if the rest of the tools are useful. If you really want to beat on your code and get as much information about its state as possible, go for PC-Lint. Of course, if you are seriously into C programming anyway, price should not be a factor. The amount of programmer time you can buy for $100—or even $500—is negligible; *never* nickel-and-dime when it comes to spending money on good tools.

NEW VERSION 2

MICROSOFT, TURBO AND MIX POWER C PROGRAMMERS...
C WINDOWS TOOLKIT™ PUTS YOU IN CHARGE OF VIDEO
and DATA-ENTRY

NEW FEATURES

- *Comprehensive data-entry and verification procedures with scanf( ) type syntax.*
- *Predefined data-entry functions for dates, SSNs, telephone numbers, currency amounts with commas, time, scientific notation, etc.*
- *Turn any window into an editor with your choice of keyboard mapping (Wordstar™ by default).*
- *Virtual windows up to available memory.*
- *Save windows to disk.*
- *Nested windows.*
- *Create and edit VGA/MCGA/EGA/Hercules fonts of any size.*

WINDOWING FUNCTIONS
- Create pop-up windows
- Create pull-down menus
- Create spreadsheet menus
- Create context-sensitive help screens
- Store windows in RAM or on disk
- Modify windows out of view
- Use 8 different types of exploding windows
- Scroll and move windows

MicroWay 386 SUPPORT*
- Runs in protected mode
- Uses 32-bit instruction set

*Extra cost option-call

SYSTEM CONTROL
- Device-independent output
- Supports VGA/MCGA/EGA/Hercules/CGA/MDA
- Detect the types of video adapter installed
- Switch between adapters
- Detect ANSI.SYS
- Control the size and position of the cursor
- Detect the enhanced keyboard
- Disable the video signal
- Delay program execution to microsecond resolution

VGA/EGA SUPPORT
- Use all 64 VGA/EGA colors
- Use VGA 29/43/50 line modes
- Use EGA 43-line mode
- Use 2 fonts simultaneously
- Design and edit custom fonts with FONTEDIT,* our font editor (included)
- Smooth scroll the screen
- Smooth pan the screen

FAST SCREEN I/O
- Write formatted output (like **printf( )**)
- Get snow-free output on the CGA
- Scroll the screen
- Use direct screen writes or the BIOS

HERCULES SUPPORT
- Create and edit your own Ramfonts™
- Detect the presence of a Hercules Card
- Detect Ramfont™ support
- Load and store Ramfonts™ to RAM or disk
- Switch between modes
- Detect whether the Hercules Card is in text or graphics mode

Full source code included — No run-time royalties.
Includes 230 page manual — 30-day Money-Back Guarantee

Magna Carta SOFTWARE

P.O. Box 475594
Garland, TX 75047-5594
FAX: (214) 226-0386

Requires: IBM PC, XT,AT, PS/2 or compatible.
Supports Microsoft C 5.0/5.1/Quick C, Borland Turbo C 1.5, Mix Power C

CALL (214) 226-6909

Only $99.95
(Texas Residents add 8% sales tax.)

Number 212 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Future of Lint

Cross-file checking of function calls was one of the biggest strengths of Lint. However, now that ANSI C has provided function prototypes, the compiler can provide such diagnosis itself. However, all is not lost, since Lint can still perform checks that few compilers are able or ever likely to do, at least for a few years. For example, Lint can check that structure, union, and enum templates are identically defined in separate modules. Likewise for macros. And don't forget that compiler vendors build and sell compilers—that's their business. Gimpel, on the other hand specializes in "quality of implementation" tools. Sure their C-terp interpreter lets you run programs, but its strength is in error-free development and debugging, as is PC-Lint's. So while some versions of Lint and some leading compiler vendors may increasingly encroach on each other's turf, I expect the gap to exist for some years yet, maybe forever.

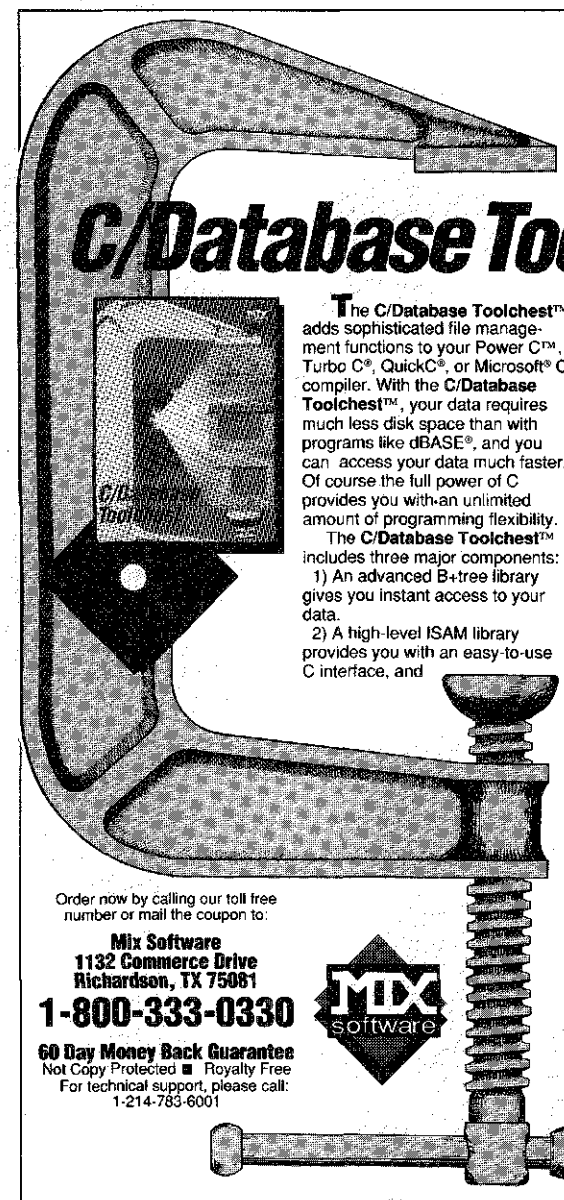
If you have never heard of or used Lint, I challenge you to put your best source through it. You may very well be unpleasantly surprised (as most people are) at the sheer volume of messages produced. Until you perform such a mechanical check on your source (and on a regular basis throughout development), your level of comfort with the quality and reliability of your code will remain significantly lower than it could be.

Rex Jaeschke is an independent computer consultant, author, and lecturer. He is also a member of the X3J11 C Standard's Committee, observer on the ISO C Standard's Committee, founding editor of The C Journal, *and the C language editor of* DEC Professional. *His new book,* Portability and the C Language, *was published by Hayden Books in October 1988.*

If you would like to comment on, or question any aspect of, the ANSI C Standard, contact Rex at 2051 Swans Neck Way, Reston, VA 22091 or via UUCP: uunet!aussie!rex.

Vendors
Gimpel Software
3207 Hogarth Lane
Collegeville, PA 19426
215-584-4261

MMC AD Systems
Box 360845
Milpitas, CA 95035
408-263-0781



Turn your favorite C compiler into a powerful database manager with the

C/Database Toolchest

The **C/Database Toolchest**™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the **C/Database Toolchest**™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.

The **C/Database Toolchest**™ includes three major components:

1) An advanced B+tree library gives you instant access to your data.

2) A high-level ISAM library provides you with an easy-to-use C interface, and

3) A complete database manager (with C source code included) shows you how to create impressive applications.

You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.

The **C/Database Toolchest**™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.

About the only thing that the **C/Database Toolchest**™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.

Now Only $19.95!

Order now by calling our toll free number or mail the coupon to:

Mix Software
1132 Commerce Drive
Richardson, TX 75081

1-800-333-0330

60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001

MIX software

Order Coupon

Name_____
Street_____
City_____
State_____ Zip_____
Telephone_____
Paying By____ Money Order____ Check
____ Visa ____ MC ____ AX ____ Disc.
Card#_____
Exp. Date_____
Disk Size____ 5¼" ____ 3½"

Qty.	Product	Price	Subtotal
____	C/Database Toolchest	$19.95	____
____	C/Database Library Source. B+tree & ISAM library source code	$10.00	____
	Add Shipping ($5 USA, $20 Foreign)		____
	Texas Residents Add 8% Sales Tax		____
	Total Amount of Your Order		____

C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

Number **266** on the Reader Service Card

JAKE™: NATURAL LANGUAGE PROCESSING USING C

Call JAKE Routines To Convert Sentences Into C Data Structures. Easy To Build Application Dictionary.

OUR CUSTOMERS USE JAKE FOR

COMMANDS Scripts
Process Control Systems
Expert Systems
Command Languages

QUERIES Hypertext Systems
Help Systems
Ad Hoc Database Queries

JAKE $495. Interactive Demo $10.
CALL (408) 438-6922 VISA, MC

EKS English Knowledge Systems, Inc.
5525 Scotts Valley Dr. Suite 22
Scotts Valley, CA 95066

Usable with Microsoft C, Turbo C, OS/2, and Windows.
Context-sensitive semantic processing.
Easy interface to any application. Mem: 64K.

Number **243** on the Reader Service Card

Free Money!

Well, almost free—
Your product can present a consistent high-quality user interface, with a minimum of effort on your part—
It's like getting paid for not working!

ScreenLib™
Complete C Source Code!
No Royalties!

Simple Screen Definition ★ Windows
Pop-up Menus ★ Context-Sensitive Help
Lots of low-level functions

Introductory Price: $69.95
Plus $5.00 shipping ($10.00 outside US)
CA residents please add sales tax

Business Computer Services
1800 S. Robertson Blvd., Suite 206
Los Angeles, CA 90035

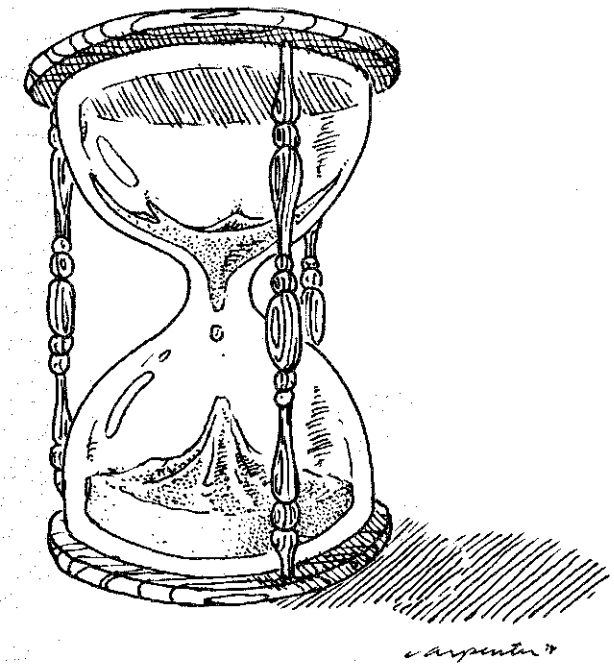
VISA/MC orders: (213) 836-5026
Demo disk available—$10.00

Number **286** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Millisecond Timer

by
Ethan Winer



When the original IBM PC was first introduced, a programmer could safely assume that if a given piece of code ran correctly on his or her own machine, it would behave the same on any other. The only possible variation was that some PCs had a standard IBM monochrome display card installed, while others had a CGA; but the basic machine was always the same.

This led to all kinds of hardware-dependent coding practices, especially when brief delays were needed in a program. For example, a BASIC programmer could use an empty **FOR/NEXT** loop, confident that the resultant delay would be the same on any PC. I still have an old book on assembly language programming that shows an appropriate count to use with the loop instruction.

When IBM released the PC-AT, these techniques could no longer be relied upon to work correctly. We've all seen those game programs that run too fast on computers that are quicker than the original PC. Indeed, TSR programs are available to slow down a PC's operation just so that it may be used with games. Now, with the current crop of PC hardware running at speeds in excess of 20 MHz (and using inherently faster processors), a new method must be found for creating hardware-independent timing and delays.

The solution I devised for my company's QuickPak Professional BASIC tools package is to create a pair of assembly language routines. The first determines the throughput of the host PC and returns a value I call a "fudge factor." The faster the host PC runs, the larger the value returned will be. The second routine uses that fudge factor as a counter and delays for a specified number of milliseconds.

The **FudgeFactor** and **Delay** routines are shown in Listing 1, and a QuickBASIC demonstration that tests their accuracy is given in Listing 2. Even though these routines are designed to be used in a QuickBASIC program, it would be trivial to adapt them to any other high-level language.

Timing the CPU

There are two ways we could calculate the fudge factor. One way is to execute a sequence of instructions a fixed number of times and then see how long that took. The other, which is the approach used here, is to execute those instructions continually until a given period of time has passed. One advantage of the latter method is that determining the fudge factor will always take the same amount of time. Another is that faster computers will produce higher, rather than lower, results.

The one timing element within every PC that can be counted on is the built-in system clock. Approximately eighteen times per second, the PC's timer executes a hardware interrupt that updates a four-byte counter in low memory. By examining the lowest byte to see when it changes, we can determine when an eighteenth of a second has passed. A BASIC representation of the **FudgeFactor** routine looks like this:

```
DEF SEG = 0                 'look in low memory
Fudge = 0                   'initialize the counter
OldTime = PEEK(&H46C)       'get the current timer byte
DO
   Fudge = Fudge + 1        'increment the counter
LOOP UNTIL PEEK(&H46C) <> OldTime   'loop until it
                                    '  changes
PRINT "This PC counted to"; Fudge   'display the
                                    '  result
```

There's one small problem with this example: Depending on where the timer is within its cycle when the loop begins, the *Fudge* variable will most likely end up being lower than it really should. Therefore, the program must wait until the timer byte in low memory has changed before beginning the actual counting loop.

Once we know how many times *Fudge* will be incremented within a fixed period, it is simple to create a delay loop that executes an appropriate number of times. One minor complication is that the *Fudge* variable must be converted to produce a delay of one millisecond. By experimentation, I determined that this value should be divided by 44 to produce the correct number of times to loop for each millisecond.

The table shows the results returned by the assembly language **FudgeFactor** function for a variety of popular PCs.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Brand/Model	Speed	CPU	Result
Original IBM PC	4.77 MHz.	8088	2003
Epson Equity	10 MHz.	8088	4228
NEC PowerMate	8 MHz.	80286	7346
Hyundai AT clone	10 MHz.	80286	9345
IBM PS/2 Model 80	16 MHz.	80386	19135
Dell System 310	20 MHz.	80386	33814

Using long delay periods (for increased resolution) and then comparing the results with BASIC's built-in **TIMER** function, these routines have proven accurate to within a few percentage points. Notice that because deriving the fudge factor will take as long as one-tenth of a second or so, this should be done once at the start of a program. The **Delay** routine may then be called as often as needed without incurring any additional delays.

The Assembly Language Subroutines

Both **FudgeFactor** and **Delay** are written to be assembled with Microsoft's MASM version 5.1. The new proc capabilities and simplified segmentation provided with MASM 5.1 make it ideal for interfacing assembly language routines with a high-level language. Let's take a look at each of these routines in turn, beginning with the **FudgeFactor** function.

The **.Model Medium** directive tells MASM that the following procedures will be treated as far, or intersegment, calls. The **.Data** instruction sets aside space for the assembly language variables, without the programmer having to specify the correct segment names for use with QuickBASIC. Within the data area are two integer variables—a low word and a high word. These are the variables that will be manipulated both when deriving the fudge factor and when performing the actual delay. The code segment is specified using the **.Code** directive, which is then followed by the two procedures.

The first step **FudgeFactor** must perform is to set the ES register for examining low memory. As long as AX is zero for that purpose, we'll use that to clear the two counter words to zero. (Assigning a memory location from a register is faster and produces less code than using a number.) Next, the current timer count is saved in the AL register. As I mentioned earlier, it is essential to wait until a new clock period has just begun to ensure that the entire period is used when counting. This is handled by the three lines of code that follow the **SyncLoop** label.

Once a new period has started, the double word counter is incremented, and the timer byte in low memory is again examined. This process continues until the timer byte has changed, meaning that one clock cycle has elapsed. The final step is to retrieve our internal counter and return it as the function's output in the DX:AX register pair.

Notice that **FudgeFactor** is designed as a BASIC function, rather than a subroutine that must be called with a passed parameter. Besides eliminating the overhead of an extra variable, using a function this way is a more natural way to return a result. Assembly language functions are not described in the documentation that comes with QuickBASIC; rather, this information is buried within the *Mixed Languages Programming Guide* that accompanies MASM 5. Assembly language functions may be used only with QuickBASIC version 4.0 or later.

Briefly, any assembly language routine may return an integer result by simply loading AX with a value before it returns to BASIC. A long-integer function (such as **FudgeFactor**) will instead use the DX:AX register combination. Floating-point functions are only a little more difficult to implement, but we'll save that discussion for another time.

The result returned by **FudgeFactor** could certainly fit into a conventional integer. However, using a long integer ensures that these routines will not be obsolete when the next—and faster—generation of computers comes along.

The **Delay** subroutine is also fairly simple to understand, although it does require two incoming passed parameters. The first is the long-integer *Fudge* variable that was assigned earlier, and the second is the desired number of milliseconds to delay. Again, by using the new MASM 5.1 proc directives, these parameters may be accessed simply by referring to them by name.

The first few lines retrieve the number of milliseconds to delay and also ensure that the value isn't zero. If the calling program inadvertently specified zero, the routine would appear to "hang" until 65,536 milliseconds had passed. Next, the incoming *Fudge* variable is retrieved and placed into the DX:AX register pair. This too is checked to be sure it isn't zero, and the value is then divided by 44 to derive the correct count value for one millisecond.

Once the appropriate value has been calculated, the program sets up two counter loops. The outer loop is executed once for each millisecond required, and the inner loop is executed (*Fudge* 44) times. At the start of each outer loop, *LoWord* and *HiWord* are initialized to begin a new one-millisecond interval. For each inner loop, these values are decremented until they pass zero.

Notice that **FudgeFactor** uses **Add** instructions for its timing determination, although **Delay** instead uses **Sub**. Fortunately, these instructions take exactly the same amount of time to execute, regardless of the type of processor.

A Practical Example

Now that we have a reliable way to create delays accurate to within a millisecond, what's it good for? Besides obvious uses such as timing games or controlling an external hardware device, a perfect application is creating visual special effects. The QuickBASIC example program shown in Listing 3 first fills the screen with a test pattern and then prompts for a number. The number you enter is then used to control the speed of an exploding "clear screen" effect.

In order to clear the screen from the inside out, another routine is needed that interfaces to the BIOS video service. This routine, shown in Listing 4, accepts values that describe any rectangular area of the screen. The color parameter allows you to specify the color to clear to, which will be 7 for normal white on black. PJ

Ethan Winer is president of Crescent Software and the author of QuickPak Professional. He may be reached at 11 Grandview Avenue, Stamford, CT 06905; by phone at 203-846-2500; on CompuServe as 72657,3070; and on MCI mail as EWINER.

Code follows.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "A Millisecond Timer" by Ethan Winer

1

LISTING 1

```
;********** Delay.Asm - two routines for implementing millisecond delays

;Copyright © 1989 Ethan Winer


;Syntax: DECLARE FUNCTION FudgeFactor&()
;        Fudge& = FudgeFactor&
;        CALL Delay(Milliseconds%, Fudge&)
;
;Where FudgeFactor was invoked earlier in the program, and Milliseconds%
;ranges from 0 to 32767.


.Model Medium, Basic
.Data
    LoWord DW ?
    HiWord DW ?

.Code

FudgeFactor Proc

    Xor  AX,AX              ;look at the low system timer byte
    Mov  ES,AX              ;through ES
    Mov  LoWord,AX          ;clear the counter variables while we're at it
    Mov  HiWord,AX
    Mov  AL,ES:[46Ch]       ;get the current timer count

SyncLoop:
    Cmp  AL,ES:[46Ch]       ;has it changed?
    Je   SyncLoop           ;no, keep checking until a new period has just begun
    Mov  AL,ES:[46Ch]       ;yes, save the new timer count

TimeLoop:
    Add  LoWord,1           ;increment our local long integer
    Adc  HiWord,0
    Cmp  AL,ES:[46Ch]       ;has the timer turned over yet?
    Je   TimeLoop           ;no, keep incrementing the variables

    Mov  AX,LoWord          ;yes, assign DX:AX for the function output
    Mov  DX,HiWord

    Ret                     ;return to BASIC

FudgeFactor Endp



Delay Proc MSecs:Ptr, Fudge:Ptr

    Mov  SI,MSecs           ;get the address for Milliseconds
    Mov  CX,[SI]            ;put it into CX for the Loop below
    Jcxz Exit               ;if they want zero, give 'em zero!

    Mov  SI,Fudge           ;get the address for the incoming fudge factor
    Mov  AX,[SI]            ;assign it to DX:AX
    Mov  DX,[SI+02]
    Or   AX,DX              ;did they forget to use FudgeFactor?
    Jz   Exit               ;yes, get out now
    Mov  BX,44              ;no, divide the fudge factor to get a loop count
    Div  BX                 ;now AX tells how many loops are needed for 1MS

Outer:
    Mov  LoWord,AX          ;set up for a 1 millisecond delay
    Mov  HiWord,0           ;use HiWord merely to match the code in FudgeFactor
```

2

```
Inner:
    Sub  LoWord,1           ;decrement HiWord:LoWord
    Sbb  HiWord,0
    Jns  Inner              ;do inner loop until Hi:Lo drops below zero
    Loop Outer              ;loop Milliseconds times

Exit:
    Ret                     ;return to BASIC

Delay Endp
End
```

LISTING 2

```
'********** Delay.Bas - demonstrates the FudgeFactor and Delay routines

'Copyright © 1989 Ethan Winer


DEFINT A-Z
DECLARE FUNCTION FudgeFactor& ()
DECLARE SUB Delay (Milliseconds, Fudge&)

CLS
Fudge& = FudgeFactor&   'call FudgeFactor first to get past its initial delay

PRINT "This PC requires a fudge factor of"; Fudge&
PRINT
INPUT "Enter the number of milliseconds to delay for: ", MSecs

T! = TIMER              'bear in mind that BASIC's timer has a resolution
Delay MSecs, Fudge&     '  of only 0.056 seconds, so for the most accurate
T! = TIMER - T!         '  comparison use a fairly long delay (>= 2000 MSecs.)

PRINT USING "BASIC says the delay was actually ##.### seconds"; T!
```

LISTING 3

```
'************ ClearScr.Bas - shows how to create an "exploding" clear screen

'Copyright © 1989 Ethan Winer


DEFINT A-Z
DECLARE SUB ClearScr (ULRow, ULCol, LRRow, LRCol, Colr)
DECLARE SUB Delay (MSecs, Fudge&)
DECLARE FUNCTION FudgeFactor& ()

CLS
PRINT STRING$(1920, "a");       'fill lines 1 through 24
Fudge& = FudgeFactor&           'get this PC's fudge factor

INPUT "How fast (1 to 10)? ", Speed
Speed = (10 - Speed) * 10       'scale, convert higher values to faster speed

FOR Row = 1 TO 11
    ClearScr 12 - Row, 34 - Row * 3, 12 + Row, 47 + Row * 3, Colr
    Delay Speed, Fudge&
NEXT
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 4

```
;********** ClearScr.Asm - clears a portion of the screen using Int 10h

;Copyright © 1989 Ethan Winer

;CALL ClearScr(ULRow%, ULCol%, LRRow%, LRCol%, Colr%)
;     range:   1-25    1-80    1-25    1-80    0-255
;

.Model Medium, Basic
.Code

ClearScr Proc ULRow:Ptr, ULCol:Ptr, LRRow:Ptr, LRCol:Ptr, Colr:Ptr

    Mov  SI,ULRow      ;get the address for ULRow%
    Mov  CH,[SI]       ;put it into CH
    Dec  CH            ;adjust 1-25 to 0-24
    Mov  SI,ULCol      ;get the address for ULCol%
    Mov  CL,[SI]       ;put it into CL
    Dec  CL            ;adjust 1-80 to 0-79
    Mov  SI,LRRow      ;and so forth
    Mov  DH,[SI]
    Dec  DH
    Mov  SI,LRCol
    Mov  DL,[SI]
    Dec  DL
    Mov  SI,Colr
    Mov  BH,[SI]
    Mov  AX,600h       ;specify "clear screen" function
    Int  10h           ;call the BIOS to do the real work
    Ret                ;return to BASIC

ClearScr Endp
End
```


PJ CODE

Code disks from six issues, only $39.95*

PROGRAMMER'S JOURNAL
CODE DISKSCRIPTION—

Subscribe Today!
You'll receive your code disks in the mail every other month throughout the year. Learn from *PJ* code or use it your own programs. Each disk contains all of the program listings published in *PJ* plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***
If purchased separately, the regular price for these six disks is $60.00.
So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!

For faster service with VISA/MC call toll-free:
800-234-0386
* Prepaid, US only. Call or write for prices outside the US.

FOR SERIOUS PROGRAMMERS ONLY

NEW RELEASE

OPUS MAKE

VERSION 5.2

The *FASTEST* most FULL-FEATURED make utility available.

Using OPUS Make will result in you spending less time doing compilations and other repetitive software tasks.

- DOS and OS/2 executables on the same disk.
- DOS version uses **only 3K of memory!**
- Read-time and Run-time conditional directives.
- Loop, Include and Error directives.
- Multiple directory support.
- Extensive macro capabilities.
- DOS version supports OS/2-like standard error redirection.
- POLYTRON PVCS, Burton Systems TLIB, LIB and SLR OPTLIB support.
- UNIX makefile syntax.
- Microsoft makefile compatibility mode.
- **Also comes with OPUS MKMF:** An automatic dependency generator that fully understands C preprocessor directives.

Both programs for only:

$89 + S&H

30-day money back guarantee.

For more information or to place an order call:

(415)664-7901

VISA/MC/cod/po

OPUS Software

1032 Irving St. #439, San Francisco, CA 94122

Serious software for programmers serious about their software.

Number **254** on the Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Exit Codes

by
Murray L. Lesser

Most "high-level" language implementations that permit linking to a private subroutine/function library can be made more useful by adding a few judicious library routines written in assembly language. The example provided here is an enhancement of the compiler implementation of Microsoft's QuickBASIC language (BC.EXE, versions 4.x) to allow compiled programs to communicate with DOS by emitting an exit code on completion. The desired exit code is passed to the subroutine as an integer argument.

QUIT.ASM (Listing 1) could be considered a "temporary TSR" module that restores itself when the calling program returns to DOS. QUIT operates by setting up an intercept for those DOS functions reached by INT 21H before jumping to the BASIC library **END** routine (B$CENP). This intercept "watches" for the DOS exit routine (4CH), merely jumping to the original DOS interrupt handler until that event occurs.

After the QuickBASIC B$CENP library routine cleans up the BASIC environment (closes all BASIC files, restores the visible cursor, and so on), it calls DOS function 4CH with a zero exit code. When QUIT intercepts this time, the original INT 21H vector is restored (using DOS function 25H), and the desired exit code is loaded into register AL before the **END** routine is completed.

A few points about "temporary TSRs" that may not be clear from the listing comments: All data used by QUIT are carried within its code segment, as (in general) the CS register is the only segment register whose contents are known when arriving at an interrupt handler. QUIT's stored data is reached with CS override instructions.

To avoid chasing its own tail in an endless loop, QUIT's cleanup routine simulates an INT 21H by first pushing the flags and then making a FAR call to the original interrupt vector. An interrupt simulation usually also includes a **CLI** instruction following the **PUSHF** to disable external interrupts until the handler re-enables them. Here, the external interrupts are never enabled in the intercept; so, the explicit **CLI** instruction is not needed.

Although the first thing DOS's INT 21H handler does is enable external interrupts, the **IRET** from this simulated interrupt will reinstate the stored flag status. Thus, external interrupts will still be disabled for the last jump to the original INT 21H vector location.

Do heed the warning on Listing 1 that QUIT will not operate correctly when fitted into a "Quick" library and used with the QuickBASIC development environment (QB.EXE). QB is not well suited for developing programs containing assembled modules. Among its deficiencies, the pre-execution "binding" of the threaded code does not check for the existence of public symbols corresponding to externals called only in Quick library routines. Since there is no B$CENP public symbol in the QB environment, the jump to nowhere will leave your system in limbo—you will have to reboot to get it back.

I tested QUIT with a compiled BASIC driver that read an input integer from the command line and emitted it as an exit code. Rather than building a batch file with a string of **if errorlevel *x* echo *x*** commands, I wrote an overdriver in Turbo C. Each cycle of an infinite loop in the C driver requested an integer input from the keyboard, spawned the BASIC test driver (providing the command-line string as **argv[1]**), and then displayed the returned exit code.

The obvious use for BASIC exit codes is with batch files, either to write menu (**ASK**) routines or to change the course of events when an uncorrected error has occurred. In this latter case, the call to QUIT must occur in an error trap that also contains a **RESUME** statement, as illustrated by the sample trap in Listing 2. If QUIT is called from an error trap with no **RESUME** statement, BASIC's error message will be displayed, including the "Hit any key" wait loop, before the program returns to DOS with the exit code. PJ

Murray L. Lesser is a retired IBM engineer. In addition to having had several articles published in the small computer press, Murray has written three books dealing with enhancing various Microsoft BASIC compilers with assembled modules. The most recent of these, Advanced QuickBASIC 4.0, *was published by Bantam last year.*

Code follows.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ CODE

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Exit Codes" by Murray L. Lesser — 1

```
LISTING 1
        PAGE    ,105
        .MODEL  MEDIUM
        TITLE   QUIT Subroutine

;                               QUIT.ASM
;    QUIT(<int>) is an assembled subroutine to provide a DOS exit
; code for programs compiled with [QuickBASIC's] BC.EXE.  All integer
; arguments greater than 255 will be masked to the low-order byte.
; After storing the argument and rerouting the DOS function INT 21H
; vector, QUIT jumps to the BASIC library END entry point (B$CENP).
; The exit code is inserted when BASIC's END routine arrives at the
; DOS "Terminate" function.
;    QUIT has been tested with BC versions 4.0, 4.00b, and 4.5.
;-------------------------------------------------------------------
;               Declare with:   DECLARE SUB QUIT (A%)
;-------------------------------------------------------------------
;    WARNING:  Do not attempt to use QUIT in a "Quick" library when
;              operating under the development environment (QB.EXE).
;-------------------------------------------------------------------
;              Written by M. L. Lesser, February 27, 1989
;                 Assembled with Microsoft MASM v 5.1

EXTRN   B$CENP:FAR              ;BASIC library END routine

.CODE

;Local data storage in code segment:
EXCODE  DW      ?               ;Store exit code
VECTOR  DD      ?               ;Store DOS's INT 21H vector

        PUBLIC  QUIT
QUIT    PROC

        PUSH    BP
        MOV     BP,SP
        MOV     BX,[BP+6]       ;Address of exit code
        MOV     AX,[BX]         ;Value of exit code
        MOV     CS:EXCODE,AX
;Get and store DOS's INT 21H vector:
        MOV     AX,3521H
        INT     21H
        MOV     WORD PTR CS:VECTOR,BX
        MOV     WORD PTR CS:VECTOR + 2, ES
;Reset INT 21H vector to QUIT's return point (CS:HERE):
        PUSH    DS              ;Save current value in DS
        PUSH    CS              ;Set DS addressability to this
        POP     DS              ;  Code Segment
        LEA     DX,HERE
        MOV     AX,2521H
        INT     21H
        POP     DS              ;Restore original value
        POP     BP
        JMP     B$CENP          ;Jump to "normal" BASIC END routine

;Return point when B$CENP uses any DOS function:
HERE:   CMP     AH,4CH          ;Is this "Terminate" function call?
        JNZ     THERE           ;If not, continue as is
        PUSH    DS              ;Save registers used before restoring
        PUSH    DX              ;    INT 21H vector
        PUSH    AX
        MOV     AX,2521H        ;Restore original vector
        LDS     DX,CS:VECTOR
        PUSHF                   ;Simulated INT 21H to avoid
        CALL    CS:VECTOR       ;  "recursion" in QUIT subroutine
        POP     AX
        POP     DX
        POP     DS
        MOV     AL,BYTE PTR CS:EXCODE           ;Insert exit code
;Continue INT 21H call from BASIC's END routine:
THERE:  JMP     CS:VECTOR
QUIT    ENDP

        END

LISTING 2
'   Sample error trap to return Microsoft's error code number as an
'   exit code:

TRAP:
   LET C% = ERR                 'Capture error code number
   RESUME DONE                  'Clear error-trapping mechanism
DONE:
   CALL QUIT (C%)
```


Multitasking

for the IBM-PC/XT/AT with

Multidos Plus

Ideal for developing process control, data acquisition, communications, and other applications which require a multitasking environment. Check these features which make Multidos Plus an unbeatable value:

- Run up to 32 programs concurrently
- Your software continues to run under PC-DOS or MS-DOS. No need to learn a new operating system.
- Use the compilers you already have. Supports software written in most compiled languages or assembly language.
- Operator commands to load and run programs, change priority, check program status, and to abort/suspend/resume task execution.
- Optional independent foreground and background displays.
- Programmatic interface through INT 15H for the following:
  - * Intertask messaging to send, receive, check or wait on 63 different message queues.
  - * Task control by means of semaphores. Get, release, check, or wait on semaphores.
  - * Change task priority.
  - * Suspend task for specified interval.
  - * Spawn and terminate external and internal tasks.
  - * Disable/enable multitasking.

Requires: IBM PC/XT/AT or true clone. MS-DOS (or PC-DOS) 2.0 or later. Enough memory to hold Multidos Plus (64KB) and all your application programs. You may also need 4 or 16KB of memory for "hidden screens" for each active task.

only: **$24.95** or

$99.95 with source code written in Lattice C and Microsoft Assembler

Call (800) 872-4566 ext. 350 for Visa and Mastercard orders only. Technical information and all other orders call (508) 651-0091 or our 24-hour FAX at (508) 655-8860. Send check or money order (Drawn on US banks only) to:

Nanosoft

13 Westfield Road, Natick, MA 01760

Mass. shipments must add 5% sales tax.
Foreign orders add $5.00 shipping & handling.
COD orders add $5.00

Number **109** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

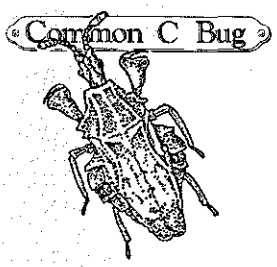
$12.00 each – includes shipping & handling

Multi-color imprints on the highest quality 50/50 cotton T-shirts.

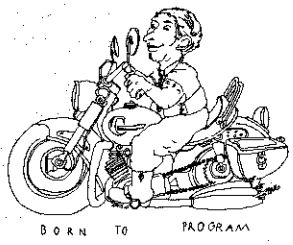
SIZES S, M, L, XL

PJ ISSUE SPECIAL!

Only $10.00! Save 17% off the regular price!



COMMON C BUG



BORN TO PROGRAM

UP-FRONT COMMUNICATIONS "SOFT-WEAR"

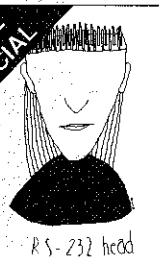
RS-232 HEAD T-SHIRT

(see design, below left)

This shirt is the correct fashion protocol for all summer communications. The popular royal blue and burgundy design is hand screened on a high-quality, platinum-colored, 50/50 T-shirt. Order today; supplies are limited. (Offer expires August 31,1989.)

PARALLEL PORTS

BUGS RULE 4 color imprint on a white shirt



RS-232 HEAD

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 5.4: Starting your own software house—technical support; UNIX books, part 3; Consulting—productivity and effectiveness; EGA set/reset capabilities; ANSI C language standard; character tables and string functions in C; MWWIND—a menu with Windows utility; variable passing and hiding in QuickBASIC 2 subprograms.

Vol. 5.5: Intel's 80386 vs. OS/2; interview with Metagraphics President Jack Davis; Consulting—making computer tools pay for themselves; ASM—the invisible CapsLock; TSRs using expanded memory with Turbo Pascal; note on optimizing Turbo Pascal; proposed ANSI C Language Standard; printer output from Turbo C programs; match-up of Turbo Basic 1.0 and QuickBASIC 3.0.

Vol. 5.6: Multics, OS/2, paging, and segmentation; The Usenix conference and other news; how to get rich in consulting; Shell in Marshal Pascal; Turbo Pascal 4.0 preview; handling fixed-length character arrays in C; the proposed ANSI C language standard.

Vol. 6.1: Consulting is an attitude; write mode 3 of the VGA; the LIM 4.0 expanded memory specification; environment variables in Turbo Pascal 4.0; a timer program in C; porting mainframe software to the PC—FORTRAN compilers under DOS and the Definicon 32-bit coprocessing engines.

Vol. 6.2: Hal Hardenbergh on RISCs, CISCs, and Fabs; Pink Noise; yet another VGA write mode; inside Turbo Pascal 4.0; the proposed ANSI C language standard; converting a Turbo C program to a TSR; 80286 and 80386 programming; 80386 software development; *The Norton Guide for BIOS/DOS Assembly.*

Vol. 6.3: Hal Hardenbergh on memory trends; Pink Noise; getting your consulting tools in order; Gordon Letwin on OS/2 memory management; virtualizing the 386; Pascal—a CASE of the jumps; Turbo Professional 4.0; BASIC—CALL, GOSUB, and multiline functions.

Vol. 6.4: Job shopping; windowing systems overview; reading VGA memory; X Window—a view from DOS; a digital clock in C for MS Windows; windowing in TopView, DESQview, and MS Windows; MS C 5.1 and the ANSI standard; a mouse unit for Turbo Pascal 4.0.

Vol. 6.5: Hal Hardenbergh on computers in the fast lane; Consulting—no-win jobs and too much work; VGA text paging; using dynamic arrays and records in Turbo Pascal 4.0; an OS/2 Presentation Manager application template; WATCOM C 6.0 and the ANSI standard; object-oriented OS/2 in C++; Ada and software engineering; REXX on the PC.

Vol. 6.6: PC DOS 4.0; OS/2 vs. UNIX; VGA screen saving and color selection; MASM quirks; addressing extra memory for DOS; DOS device drivers debugging and information utility; proposed ANSI C language standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating-point hardware and software.

Vol. 7.1: GRAPHICS: High-resolution VGA color modes; graphics processors; modifying type constants stored in Turbo Pascal 4.0 and 5.0 compiled EXE code; comparing Pascal overlay managers; JPI's Top-Speed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; Lattice C V3.3; smart terminal emulations in Windows; the COBOL-85 ANSI standard; Pink Noise—industry news and reviews.

Vol. 7.2: OPERATING SYSTEMS: undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; how to become a reseller; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book.*

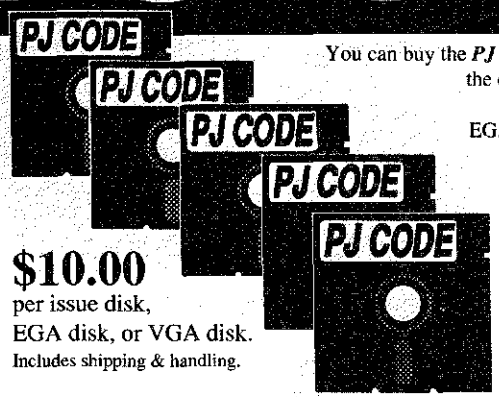
Vol. 7.3: DATABASE JUNGLE: neural nets; fast line drawing for the EGA and VGA; embedded SQL for a relational database; Btrieve; converting database files using filters; the proposed ANSI C language standard; a substitute for the missing TYPE command in Windows; user interface tools for DOS; reviews of dBASE C libraries, software tools, FORTRAN tools, conferences, exhibits, and books.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00
per issue disk,
EGA disk, or VGA disk.
Includes shipping & handling.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

Diskettes are available for the folowing issues:

5.5	6.4	7.3
5.6	6.5	7.4
6.1	6.6	EGA
6.2	7.1	VGA
6.3	7.2	

360K MS-DOS disks standard; other formats available.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Software Engineering Store

C

Borland Turbo C v2	$105
Borland Turbo C v2 Prof.	$175
Blaise C Tools Plus/5.0	$99
Creative Vitamin C	$169
Gimpel PC Lint	$99
Microsoft C v5.1	$325
Microsoft Quick C	$73
Novell Btrieve	$199
Softfocus Btree/ISAM comprehensive file/index manager with all source	$99
Wordcraft C Workshop C training manual & software	$69
Zortech C++ Optimizing C++ Compiler	$129

Pascal

Borland Turbo Pascal v5.5	New!	$105
Borland Turbo Pascal Prof.	New!	$175
Microsoft Quick Pascal	New!	$73
Turbo Power Professional v5		$99

Other Languages & Tools

Digitalk Smalltalk/286	$164
Borland Assembler & Debugger	$105
Copy Tech Disk Duplicator Pro	$129
Matrix Layout	$129
Microsoft Macro Assembler	$111
Microsoft Windows SDK v2.1	$349
Mortice Kern MKS Toolkit v2.3 KORN shell & 130 UNIX™ -like utilities	$175
Periscope I 512K v4 New!	$599
Periscope III v4	$1199
Polytron Professional PVCS	$359
Qualitas 386-Max 80386 memory manager	$69
Solution Systems Brief	$call
Whitewater ACTOR High productivity development for MS Windows	$449

386 Hardware

Hauppauge 386XT replaces your XT motherboard to provide top 80386 performance with maximum compatibility. The 386XT works with virtually all 8-bit and 16-bit adapter cards and runs standard versions of 16-bit and 32-bit software including DESQview, OS/2, Windows/386 and UNIX. The board includes 16Mhz 32-bit CPU, 1MB zero wait state memory, 80387 socket and 8 slots (2 16-bit, 1 32-bit)

The 386XT is backed by a 30 day money-back guarantee and one year warranty.

386XT w/1MB RAM	$1395
20Mhz AT w/1Mb	$1695
33Mhz AT w/4MB	$4295

Hauppauge!

Multi-Tasking

DESQview *by Quarterdeck* is the only multi-tasking environment that provides complete compatibility with the XT, AT, 386 and PS/2 and also fully exploits the 80286 and 80386. $99

QEMM	$55
API Reference	$54
API Ref & C Library	$179
API Debugger	$119
API Panel Designer	$119

Debugging

Periscope v4 *by Periscope*: The Periscope debuggers provide the combinations of hardware and software you need to solve really tough programming problems. Periscope includes all the debugging features you need, but with minimum overhead and maximum flexibility. With Periscope you can debug programs which need all memory, have real-time requirements and/or are picky about vector usage.

Periscope I w/512K 0k memory overhead!	$599
Periscope III hardware breakpoints and trace-back for real-time applications	$1199

The Software Engineering Store

1-800-443-7176

In Calif: 1-800-635-0969

(415)957-0111

Visa • MasterCard
COD • Check

- Most orders shipped within 24 hours
- Free shipping on orders over $100

575 Mission Street, #600-PJ
San Francisco, CA 94105

Free Catalog & Newsletter

We carry hundreds of products from dozens of publishers. Call or write for your free subscription to our catalog and newsletter.

Returns

We want you to pay for only the tools you really use. If you order anything featured in this ad (or most of the products in our catalog) you can open and use the product for up to 30 days and still return it if it doesn't meet your needs.

The Software Family has a new name!

We've changed our name, but not our ownership, staff or outstanding service. The Software Engineering Store continues our tradition of fair pricing, expert technical support and honest business practices.

We're Moving!

The Software Engineering Store will be moving on August 1, 1989.

Our new store will feature:

- enlarged display area
- training facilities
- improved demo facilities

New Address

The Software Engineering Store
657 Mission Street, Third Floor
San Francisco, CA 94105

Phone numbers are unchanged.

Prices subject to change without notice - ©1988 TSE. All Rights Reserved • Jul/Aug '89 PJ

Number 99 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Communications

RamNet
Software Concepts Design
594 Third Avenue
New York, NY 10016
212-889-6431

RamNet version 3.44 ($149) is an interesting PC communications and networking product from Software Concepts Design. The package provides a RAM-resident (TSR) background communications facility on the PC. A second unusual feature is the capability to support UUCP networking with other UNIX machines. RamNet allows the user to perform file transfers as well as run a bulletin board system in the background. The package supports script files and E-mail and can even function as a Telex machine.

The software consists of a main RAM-resident communications engine along with a family of menu-driven and command-line programs to interact with the TSR-based communication engine. The RAM-resident engine can also be invoked with a hot key. An applications program interface is also provided with C source code for writing your own software to interface with the TSR-based RamNet engine, although this is not necessary for normal use.

File transfers using XMODEM (both checksum and CRC) and UUCP protocol "g" are currently supported. Other protocols are to be added in the next version. The software can be configured for both direct-connect lines and modems at speeds of up to 19,200 baud. Two serial ports (COM1 and COM2) are supported. Communications with another system can be scheduled to occur at a particular day and time. The background BBS supports logins, passwords, privilege levels, and other security features. The BBS is also designed to communicate automatically with other RamNet sites for scheduled E-mail and file transfers. Software Concepts maintains a RamNet BBS, which is available 24 hours a day for product registration, updates, information, and communicating with other RamNet sites. RamNet fully supports UUCP (UNIX-to-UNIX CoPy) protocols and allows networking (mail, file transfer, and remote execution) with UNIX machines.

The software is configured with a menu-driven INSTALL utility for machine specifics, modems, and adding systems or users you plan to communicate with. The basic RamNet engine uses about 135K and depends on access to the timer interrupt to occasionally steal CPU time from the foreground task for file I/O and printing. RamMenu, a menu-driven program interface, provides access to the RAM-resident portion along with on-line status and debugging information for communication activities. Nine levels of debugging activity are available; these can be logged to a file, a printer, or the screen.

RamNet comes with a 190-page manual bursting at the seams, about one-third of which are appendices (no index is included). The information on installation and setup is a bit brief (particularly for UUCP), but I was able to muddle through it. A new manual is in the works. The RamMenu utility is relatively easy to use but requires some understanding of all of the RamNet options. Once the TSR is installed, it can be removed only by rebooting the system. I was able easily to set up a UUCP connection with the local university and schedule file transfers and E-mail. The current version of UUCP packet "g" implemented uses a sliding window of 1 (basically, no sliding window), so its use over packet-switched networks would suffer. A revision is to correct this weakness. Overall, I found RamNet to be a feature-rich background communications package that also supports UUCP. RamNet comes with a 30-day, money-back guarantee.

DOS Extenders

DOS/16M
Rational Systems
P.O. Box 480
Natick, MA 01780
508-655-2753

Rational Systems has released version 3.25 of *DOS/16M*, their "DOS extender" for AT-class machines. *DOS/16M* provides a way to give DOS programs access to up to 16Mb of extended memory on 80286 or 80836 machines. *DOS/16M* is a 286 DOS extender that functions by putting the CPU into the protected mode and providing an interface for your programs that looks almost identical to DOS. All documented DOS and ROM BIOS interrupts are supported. *DOS/16M* handles DOS and BIOS interrupts by switching back to the real mode and translating addresses (and copying data if necessary) to make the actual DOS or BIOS calls. This provides developers with a relatively easy port to the larger memory space under DOS without changing to a new operating system. Applications based on *DOS/16M* load just like other DOS applications and are completely transparent to users. Commercial products using *DOS/16M* have been developed by the likes of Autodesk (AutoCad), Gupta (SQL), Informix, and most recently Lotus.

In order to use *DOS/16M*, a developer would need only to relink already compiled code (Large or Huge model) with some library code supplied by Rational. But most likely, your source code would also be recompiled with options to generate 80286 instructions for better performance. *DOS/16M* is compatible with a large list of compilers, including Microsoft C and FORTRAN, WATCOM C, Lattice C, Aztec C, and MetaWare's High-C and Pro-Pascal. Assembly language can also be used, but take care to ensure that your code will run in the protected mode. This means not using segment registers as scratch registers and generally maintaining code and data in separate segments. Compatible linkers include Microsoft, Phoenix, Lattice and Phar-Lap. I also tried SLR's OPTLINK, which seemed to worked fine.

This product is supplied with a number of development utilities, object files (and some source) for supported compilers, and object files (with some customizable source) to the *DOS/16M* DOS extender run-time code itself. If your program uses undocumented DOS or BIOS interrupts, the *DOS/16M*-supplied library can be customized to support such special features. A powerful protected-mode debugger (*Instant-D*) is supplied for testing and debugging protected mode applications. *Instant-D* fully supports CodeView debugging information and provides a multiple window environment. Debugging windows simultaneously display source code (optional assembly code), registers, data, and stack, and provide a command window. The development process uses a MAKEPM utility that modifies the normal DOS EXE file linker output for operation in the protected mode. Other utilities include a LOADER program used during development and a SPLICE utility that combines your debugged protected mode program with the LOADER kernel, creating a self-contained, deliverable EXE file. Several excellent diagnostic and test utilities are also provided for determining hardware compatibility and the speed of switching between real and protected modes.

The *DOS/16M* run time is tightly engineered, using only 24K of DOS memory (part of the run-time code is placed in extended memory). This leaves DOS memory available for networking software, TSRs, and such. The run-time also supports extended features for allocating either DOS or extended memory, supports bimodal (real

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Application Programmers:

AXOS a secure, multi-language, multi-user, full, flexible, indexed file manager for only $395.

BASIC • COBOL • C • FORTRAN • PASCAL • ASSEMBLER

"AXOS provides 57 functions, including audit trail and key-only file management functions, and 64 index paths versus Btrieve's 31 operations and 24 access paths." *–Victor E. Wright, PC Tech Journal 10/88*

AXOS is a fast, full-featured, key-indexed file management system designed to provide application programmers and software developers a flexible and versatile data storage and retrieval mechanism with the least amount of overhead. It can be used in either single or multi-user LAN environments on the IBM PC PS/2 and compatibles. As AXOS can be called from any programming language through a common communication protocol, it allows one to manipulate data in a manner not possible with other DBMSs or access methods. AXOS also offers a full complement of file management functions, system management functions, productivity tools, and debugging aids designed to ensure easy and accurate development and support of microcomputer applications systems. No royalties are charged for your programs that use AXOS routines.

FEATURES

- SECURITY: LAN computer name and password protection. Eight types of file I/O security by computer.
- AUDIT: Complete (update and retrieval) or partial (retrieval) capture of all AXOS activity, available with selection criteria both online or hardcopy.
- DATA INTEGRITY: Ensures full data integrity in the event of a program, system, or power failure.
- INTERACTIVE MONITOR: Instantaneous interactive monitoring of AXOS activity as a program is executing.
- FILE CONVERSION: Convert or merge existing files of other types into any one of the AXOS file formats available.
- FILE LIST: Produces a formatted report of the contents of all AXOS file types and a variety of DOS sequential file types.
- PROGRAM INVOKED ERROR ROUTINE: Used by application programs for processing unexpected I/O return codes.
- RECORD ENCRYPTION AND COMPRESSION: All records contained on AXOS files are encrypted and compressed so as to reduce disk storage requirements.
- PAGE BUFFERING: AXOS builds and manages all of its own buffer space so as to minimize disk I/O and decrease processing time in both single user or multi-user (LAN) environments.
- PARAMETERS: All of the AXOS keyboard commands utilize parameters entered at the time that the command is executed. This provides a level of flexibility wherein a command may be tailored to meet specific user requirements.

SPECIFICATIONS

- Languages supported:
  BASIC COMPILED BASIC COBOL
  FORTRAN C ASSEMBLER PASCAL
- Local Area Networks Supported:
  IBM PC NET AT&T StarLAN
  3Com 3 + Share Novell Netware
- File Types Supported .....: Sequential
  Single Index
  Mutli Index (cluster)
- Maximum Indexes .....: 64 per file
- Maximum Key Length .....: 240 bytes for unique keys
  236 for non-unique keys
- Maximum Record Size .....: 64,000 bytes
- Maximum Records Per File .....: 16, 777, 216
- File Capacity .....: 4.2 billion bytes
- Maximum Files & Indexes
  allowed open at one time .....: 253
- Operating System Requirement .....: DOS 2.1 or above
  128–256KB of memory

McSoft INCORPORATED™

P.O. Box 12044, Atlanta, GA 30355-2044

Try single- and multi-user, multi-language AXOS with 30 days free support. Call

404-984-0608

Number **230** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and protected mode) interrupt handlers, and provides compatibility with EMS 3.2 and 4.0 drivers, VDISK (DOS), RAMDRIVE (Microsoft), DESQview, disk caches, and TSR programs. The run time also supports the Virtual Control Program Interface (VCPI) developed jointly by PharLap and DESQView and supported by most 80386 control programs. Example programs are even included to demonstrate how to port TSR programs to *DOS/16M* in such a way that most of the program is in extended memory. In general, small programs may run a little slower in the protected mode. But if overlays or virtualizing are required under DOS, then the benefits of the larger protected-mode address space can be impressive indeed.

The package comes with an well-written 200-plus-page manual with a decent index. The manual covers compatibility, converting to the protected mode, *DOS/16M* extended functions, utility programs, and the debugger. Very detailed examples are provided on how to write a bimodal (real and protected modes) interrupt handler, how to create TSRs that mostly reside in extended memory, and how to use EMS and other features. A tutorial describes how to convert your program to the protected mode using the supported compilers and linkers. The initial investment required for development is steeper than its competitors (A.I. Architects), but the average run-time cost is comparable. *DOS/16M* is sold with an Initial Development License for $5,000 which includes free distribution for the first 200 EXE files (including the run time). License fees for additional copies vary from $30 down to a low of $5 per copy, based on volume.

Using *DOS/16M*, I was able to convert several large text-mode applications written in C to run in the protected mode in less than an hour each. As a second test, I used the Blaise Asynch Manager package as a starting point for a bimodal serial interrupt handler, and within several hours, I had converted their ASM code for use in the protected mode. The modest DOS overhead is impressive indeed. If your DOS program is getting cramped, this is a truly powerful and attractive product for freeing applications from the DOS 640K limit with little effort.

Graphics and Animation

PCX *Technical Reference Manual*
ZSoft Corporation
450 Franklin Rd., Suite 100
Marietta, GA 30067
404-428-0008

The PCX file format developed by ZSoft for their PC Paintbrush products has clearly become established as a standard graphics file format on the PC. Various versions of ZSoft's PC Paintbrush have traditionally been bundled with mice from both Microsoft and Logitech. Not only are PCX files supported by most major programs (the likes of Ventura and PageMaker), but the PCX image file format has also become a de facto standard used by most PC scanning software. Most major PC fax boards also directly support PCX image files. The PCX image file format consists basically of a header followed with scan line data compressed using run-length encoding.

This short (21 pages) *Technical Reference Manual* is available free from ZSoft. The manual briefly documents the PCX image file format, PC Paintbrush bit-map font format, and technical information (function calls, errors, etc.) on FRIEZE, ZSoft's memory-resident screen-capture and printing utility. Some simple routines in C are provided to read data from PCX files. Although this manual is the latest from ZSoft (dated 1988) and newer than the version I described several years ago in this column, ZSoft's own documentation tends to lag behind the actual PCX format changes. The ZSoft manual also strongly recommends the PCX Programmer's Toolkit from Genus (see below).

PCX Programmer's Toolkit
GENUS Microprogramming
11315 Meadow Lake
Houston, TX 77077
800-227-0918 or 713-771-4914

GENUS Microprogramming has released version 3.52 of their PCX Programmer's Toolkit. The Toolkit consists of an extensive library of functions (62 routines) for manipulating PCX files along with nine useful PCX utility programs. The Toolkit functions are written entirely in assembly language providing small, fast routines for working with PCX files. The Toolkit supports the display modes of the Hercules, CGA, EGA, MCGA, VGA, and extended VGA display adapters and comes with a comprehensive set of compiler interfaces. Linkable libraries and include files are provided for Microsoft C, QuickC, Turbo C, Lattice C, Microsoft Basic, QuickBASIC, Microsoft Pascal, Turbo Pascal, Microsoft FORTRAN, Clipper, and assembly language. Small, Medium, and Large model libraries are included.

The Toolkit functions and utilities currently support 21 different video modes, including 800x600 256-color extended VGA modes. The utility programs were built with the toolkit and illustrate the speed and comprehensive nature of the toolkit functions. PCX utilities include pcxGrab (a TSR screen-capture program), pcxShow (a PCX-file displaying utility), pcxPrint (a PCX printing utility for two color images to HP LaserJet printers), pcxLib (a PCX library utility to archive multiple image files), and pcxCut (a cut utility to create cutouts or windows from PCX files). The pcxGrab program will capture both text and graphics screens. Text screens can be converted to PCX image files (your choice of target video mode) using a pcxTrans program. Other useful utilities allow changing from an older PCX format to the current file format, displaying the header information of a PCX image file (original display resolution, image depth and width, palette, etc.), and a utility that allows locating pixel coordinates by displaying an image with a moveable cross-hair cursor. The utilities are all small and fast and support all of the Toolkit display modes. The Toolkit will function in as little as 128K of memory.

The more than 60 Toolkit functions supplied can be organized in the following categories: buffers, files, display, virtual screens, text screens, palettes, printing, PCX headers, video, and miscellaneous. The functions support using expanded memory (LIM 4.0) for storing image buffers. The functions are space efficient and will by default use a small, 1000-byte buffer for file display, printing, and functions requiring intermediate buffers. However, allocating a larger buffer will provide faster file-display operations. The concept of a virtual buffer is integrated into the Toolkit. Virtual buffers allow a programmer to work with PCX image files larger than the available screen-display resolution (such as fax or scanned files), via scrolling and paging. Virtual buffers for uncompressed images can be allocated in EMS or conventional memory for even faster display operations. One Toolkit function will even try to identify the type of VGA chip sets—Tseng, Paradise, or Video 7. The Toolkit functions will also work with other graphics libraries (Microsoft and Borland). The functions appear well crafted, with the programmer in mind. To change a program for a different display resolution (video mode), usually only one Toolkit function call need be changed.

Documentation consists of a professional 300-plus-page manual. Various chapters cover getting starting, installation, the PCX utilities, the PCX programming interface, a complete PCX function reference, applications, and examples. The chapter on the PCX interface describes the interface and how features are handled for the different languages. The reference section is excellent and consists of two pages for each function with a description, return value, related functions, summary, and examples for each supported language. Appendices cover Toolkit Constants, display types supported, return codes, printer parameters, structures used, utility error messages, and the PCX file format. The Appendix PCX file format and header information is more up-to-date than the ZSoft documentation. The manual has a comprehensive table of contents and index for finding information fast.

The PCX Programmer's Toolkit is available for $195 (with full source code, $495). There are no royalties for use of the Toolkit functions. GENUS also offers two other packages: PCX Text for bit-mapped text ($99) and PCX Effects for adding special effects ($99). Using the Toolkit functions, I was quickly able to incorporate code for grabbing and saving screen images in PCX format. Although the Toolkit doesn't currently support the old AT&T 6300

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

640x400x2 resolution mode, I was easily able to modify a few source functions and add it. Overall, I found this Toolkit to be an exceptional product, well worth its price.

GRASP 3.1
Paul Mace Software
400 Williamson Way
Ashland, OR 97520
503-488-0224

Paul Mace Software has released version 3.1B of GRASP, their GRaphic Animation and presentation System for Professionals ($149). GRASP is a tool designed especially for creating and running animated graphics demonstrations and interactive tutorials and for adding visual effects to presentations. The GRASP system consists of a graphics animation language interpreter (GRASP), a paint program (PICTOR), a screen-capture and save utility (CAP), a graphics screen print utility (INSET), a librarian, and a show picture utility. The current version supports 20 video modes, including CGA, Hercules, AT&T mono, EGA, MCGA, VGA, and some of the extended VGA modes. This package is very different from the PCX Toolkit described above, the fundamental difference being that GRASP is an interpreted graphic animation language.

The GRASP input language is a text file of keywords and commands that is interpreted at run time. The GRASP language includes about 74 commands and supports creating macros from existing commands. The language supports a wealth of graphic commands, along with flow and looping constructs found in other languages. The primary animation commands are PFADE (picture) and CFADE (clipping) which support 25 different fades on graphics screens. These fades include left to right, edge to center, top to bottom, screen halves, vertical quarters, finger wipes, filter wipes, sparkle fades (random), diagonal, aperture dissolve (edge to center and center to edge), clockwise, double slant, and more. The development version of GRASP provides a text editor for creating input files as well as an overlay to invoke the PICTOR paint program. GLIB, a graphics library utility, allows archiving of PICTOR files, clippings, and GRASP input text into a single library file for distribution. A run-time only version of the interpreter is also included for distribution with your GRASP input files.

GRASP also supports a simple interface from other languages. Sample interfaces are provided for Microsoft C, QuickC, Turbo C, Turbo BASIC, QuickBASIC, and assembly language. This interface requires that GRASP already be memory resident. Your application makes calls to the GRASP by passing a line to be interpreted as a string to GRASP via an INT 10H interface. GRASP uses about 230K when EXECing another application. The PICTOR utility uses the PIC file format used by PC Paint program (not to be confused with PC Paintbrush). PICTOR is similar to PC Paintbrush but is not so polished a product, with some limited features like line width.

The GRASP manual consists of a section on the GRASP language, a section on PICTOR, and a short utilities manual (no index). Each GRASP language element is detailed on 2+ pages with a sample use of the command. Several appendices document the video modes supported by both GRASP and PICTOR. The manual could benefit from a extended tutorial or overview on how to use the GRASP language effectively.

The GRASP interpreter must expand the PIC file to an image format before it can apply its animation effects. This requires large buffers particularly for the extended VGA modes. The GRASP utility programs are functional but not as polished as the PCX Toolkit. Being interpreted, the GRASP package allows for quick development and testing of an idea (once one learns the language proficiently). However, it would be very attractive if a library of equivalent functions were available for linking in with a developed application (foregoing the interpreter) after the demo phase. Be forewarned that the graphics library file for a decent demo can get pretty large (larger than the space on a 1.2Mb disk). As a tool for developing attractive presentations, GRASP does an effective job.

File Managers

AXOS
McSoft Incorporated
P.O. Box 12044
Atlanta, GA 30355-2044
404-984-0608

McSoft has released version 3.31 of AXOS—Advanced Index Organization System ($395). AXOS is an indexed data and file manager designed to be used by PC application programmers. Similar to Btrieve (Novell), AXOS can be used to build simple or very sophisticated database applications. In contrast with Btrieve, the same version of AXOS supports both single-user and multi-user network applications. AXOS consists of a main AXOS TSR file manager engine, a series of utility programs to work with the AXOS engine, and a set of include files and source and object libraries for interfacing with the AXOS engine. Linkable libraries are provided for Microsoft C, COBOL, FORTRAN, Pascal, Realia COBOL, and assembly language. Assembly language source code is provided for these interfaces. The primary interface uses the Pascal calling conventions and could also be used with other C (Borland, WATCOM, MetaWare, etc.) and Pascal compilers supporting Pascal calls. One standard interface is provided for use with all of the above programming languages except BASIC. Both near and far calling conventions (and data pointers) are supported.

AXOS supports three types of file structures—AXOS Sequential, AXOS Indexed, and AXOS Index Cluster. AXOS Sequential files use a "relative record number" (4 bytes) for access. AXOS Indexed files support a single, unique key field, which may be up to 240 characters in length. AXOS Index Cluster files are the most versatile. Up to 64 different keys (called "Index Paths" by AXOS) can be identified for each record. Each Index Path can be up to 240 bytes long and contain any number of contiguous and noncontiguous fields extracted from a record. This file type also supports *sparsing strings*, which if encountered when building an index, will be omitted (for example, zero values). All three file types use a three-tier, hierarchical indexing structure designed to allow fast record access. Records are limited to 16Mb per file. All records are of variable length up to 64,000 bytes. Each level of hierarchy is made up of one or more index blocks, with each block containing 256 pointers to subordinate blocks. At the lowest level of the hierarchy, the pointers point to actual file records. Normally, all records written to AXOS files are encrypted and compressed for security and to save disk space. Typical compression saves about 50 percent of the space for a given record.

When AXOS is loaded (as a TSR), it becomes an integral part of the operating system. The AXOS TSR engine is reasonably memory efficient, using 66K. The RAM-resident core has some interesting features. If you use the DOS DEL command or another program to delete an AXOS file, the AXOS engine will warn you before allowing the erasure to take place. AXOS may also be unloaded from memory, using one of the supplied utilities, when not needed.

AXOS supports more than 60 functions for file management. These functions are invoked using a Function Control Table passed to the AXOS TSR engine. The details of this actual mechanism are partly hidden by the higher language interfaces. Functions are provided for the typical file (open, read, delete, close) and record (get, insert, replace, delete, find) operations. Powerful functions are available for beginning and ending a transaction, backing out of a transaction, turning an audit log on and off, turning an AXOS monitor (system monitor and debugging facility on and off), and holding and freeing a record. AXOS will maintain a complete audit file of AXOS functions invoked and transactions.

Almost 20 supporting modules and utility programs are also provided. A RECOVER feature allows recovering (to another machine on the network) from corrupted files, based on a LOG file being intact. This prevents power failures from permanently damaging AXOS files. Modules can backup and restore AXOS files, control AUDIT files, control the system MONITOR function, provide ACCESS to other network machines, created formatted reports of AXOS files and activity, and reorganize and CONVERT files of varying structures to and from AXOS files. Other modules provide security, network, testing, and tuning activities.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The manual consists of a solid 500-plus-page manual providing an introduction, detailed AXOS function and command descriptions, information on installation and operation, and documentation for the language interfaces. Each AXOS function is described separately and fully, including required fields, returned fields, and effects on position in file. The command-module reference describes in detail each AXOS utility command and its effect. Appendices document AXOS file structures, provide sample AXOS reports, and include sample program listings in various languages (also on disk). No index is provided.

This package is solid but could definitely use more examples and tutorial material illustrating its proper usage. The single example program (in various languages) was only provided as source (no executable code). Under my own testing, I found the MONITOR option to be a very useful debugging tool for locating where an error occurred. The MONITOR option will display a screen message for each function invoked with parameters supplied and the return status. Overall, I found the AXOS package to execute quickly and provide some very interesting and useful features for file management.

Books

Microsoft Mouse Programmer's Reference
Microsoft Press
P.O. Box 97017
Redmond, WA 98073-9717

Microsoft Press recently decided to publish more of the Microsoft reference manuals as professional books. The *Microsoft Mouse Programmer's Reference* is one of the first of these efforts. Judging by example, this arrangement will be fruitful for programmers. This particular book is well laid out and attractively presented. Also included are two companion 5.25 disks with sample mouse menus, mouse and EGA libraries, and programming examples illustrated in the text.

The *Microsoft Mouse Programmer's Reference* provides both an overview and a technical resource for programmers. The book is divided into four sections. An introduction provides an interesting history of Microsoft mouse hardware and software drivers. The following section covers creating "Mouse Menus" using a mouse menu programming language and the supplied MAKEMENU and MENU disk programs. The core of the book covers the mouse programming interface and documents the mouse functions in detail. The meanings of terms such as *virtual screens*, *mickeys*, and *mouse sensitivity* are explained well. All functions from Reset/Status (function 0) to Get Version (function 36) are described. Programming examples in assembly language, C, QuickBASIC, and interpreted BASIC are provided for invoking each mouse function.

The programming interface to EGA.LIB (the Microsoft EGA Register Interface) is also detailed, with notes warning programmers of potential problem areas. A chapter provides some more complete test and demo programs illustrating use of the mouse interface. These examples also include sample code in FORTRAN and Pascal. The appendices document the command-line options for both the Microsoft mouse driver and the CPANEL (control panel) utility, error and menu messages, interfacing with Borland's Turbo Pascal, using Hercules graphics cards, the PC character set, and keyboard scan codes. A 12-page index provides easy access to subjects.

The final question you're probably asking is whether *all* the Microsoft mouse functions are fully documented. The answer is: almost. Function 45 ('M') returns a pointer to the Microsoft copyright, and function 77 ('m') returns a pointer to the version number. If you need to do some mousing around on the PC, this well-written book might just click for you. PJ

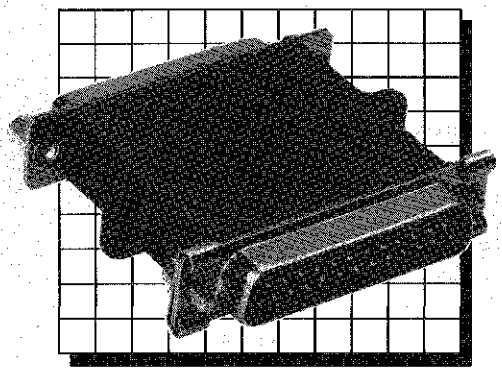
PROTECT INTELLIGENCE WITH INTELLIGENCE

PROTECH KEYs:
- are totally transparent to the end user
- allow unlimited back up copies
- free up disk drives
- do not interfere with peripheral operations
- are easy to install
- protect software developers' revenues
- provide the highest level of protection available

PROTECH'S KEYs provide the highest level of protection because of:
- assembler based routines (not drivers)
- encrypted routines
- physically unique hardware KEYs
- sophisticated software installation advice

THE KEY
The KEY provides effective software protection while insuring customer satisfaction. The KEY is a random response device which is designed for identically reproduced software packages.

SOFTWARE PROTECTION KEYs BY PROTECH



WE HAVE THE *KEY* TO SOFTWARE COPY PROTECTION

For a demonstration package or additional information, please write or call:

1-800-843-0413

THE MEMORY KEY
The MEMORY KEY is a programmable software protection device. Each byte of memory can be addressed in groups or individually. Possible applications for the MEMORY KEY include:
- modular package control
- serialization
- customization
- access control to PCs

Information stored in the MEMORY KEY can also be altered by your application software in the field. This dynamic writing to the MEMORY KEY opens up unlimited possibilities. Possible applications for the dynamic read/write function include:
- demo control
- software leasing
- updating modules in the field
- any "counter" operation

PROTECH MARKETING, INC.
1804 W. Southern Parkway • Building A-112
Durham, North Carolina 27707
(919) 490-4970 FAX (919) 490-4974

Number 228 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

UNDERSTANDING C VIDEO TAPE

Get guaranteed results with this three volume video course! For expert or novice—these tapes (total 6 hours) cover ANSI C for today's professional programmer. Loaded with examples, these tapes cover it all! Handy reference book included.

$59.95 postpaid. Add appropriate tax in TX. Send check or money order to:

AUTOMATION EDUCATION
2309 Royce Drive
Arlington, TX 76016-1232
817-496-8747 **RS# 207**

YOUR CUSTOMERS DESERVE THE BEST - CUT YOUR SUPPORT COSTS

Install your software as if you were there yourself, adapting to hardware, software, environment and installer wishes. Just "fill in the blanks" to create multiple adaptable options in minutes, intelligently controlling AUTOEXEC, CONFIG and much more. New installations, updates, MODS. 250-page User's Guide, 45 pages on automatic handling of special and abnormal situations.

INSTALLATION TOOLKIT ver. 3.0, $99.00. No copy protection/royalties, money-back guarantee, VISA/MC.

Helpful Programs, Inc.
P.O. Box 16078
Huntsville, AL 35802-1627
800-448-4154 **RS# 280**

DOCUMENTATION COMPILER

EASILY CREATE AND SHARE SOURCE CODE DOCUMENTATION. DocuMotion builds cross-indexed project documentation libraries from your source code comments. Language-independent. Resident/nonresident view utility gives instant shared access to all project documentation, manuals, etc. powerful print and copy capabilities. Folding documents, electronic bookmarks, search, LAN-compatible, more.

Personal version $149, 3 station network $359, 6 station network $597.

NWP-Intelligent Solutions Inc.
P.O. Box 20478
Bloomington, MN 55420-0478
612-884-5860 **RS# 283**

LOW-COST ADVERTISING

PJ Billboard ads are a very affordable way to reach potential customers. Text ads include a headline, company name, address, and phone number, Reader Service number, and approximately 400 characters of sales pitch. Display ads are 1 5/8" wide by 2 3/8" deep. Costs are $125 for a single insertion, $105 each for 3 insertions, and $95 each for 6 insertions. Ads must be prepaid and are non-commissionable. Call the ad department for deadlines.

Programmer's Journal
P.O. Box 30160
Eugene, OR 97403
503-747-0800
Fax: 503-746-0071

PJ ISSUE SPECIAL

RS-232 HEAD T-Shirt
ONLY $10.00
SAVE 17% off the regular price
See page 88 to order.

PJ CODE IS AVAILABLE

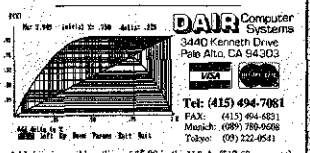
ON DISK—CALL
800-234-0386
$10.00 each, VISA/MC

See ads on pages 8 and 88 to order or see pages 58 and 62 for information on BBS downloads.

CHAOS Manner

CHAOS Manner is an exciting introduction to the new science of chaos. It introduces important concepts without using non-linear differential equations.

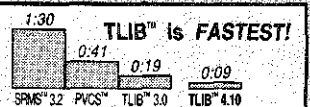
- Powerful, interactive graphics
- Spreadsheet-type user interface
- 256x magnification facility
- Course and fine display modes
- Rigorous treatment without the math
- Discusses chaos in neural nets
- Complete source code included
- Fast PL/D compiler, 90+ page manual
- Print graphics with DOS utilities
- Tutorial manual
- Runs on ordinary PC **$97.95**



DAIR Computer Systems
3440 Kenneth Drive
Palo Alto, CA 94303
Tel: (415) 494-7081
FAX: (415) 494-6831
Munich: (089) 780-9686
Tokyo: (03) 222-0541

Add shipping and handling of $5.00 in the U.S.A. ($12.50 overseas) and sales tax in California. Needs PC, AT, or PS/2, or compatible; 256K; DOS 2.0+; CGA, EGA, VGA, or compatible adapter.

Number 5 on the Reader Service Card



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. SRMS and TLIB 4.10 are later versions.

TLIB™ is FASTEST!

TLIB™ is BEST!

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, Sperry. Integrated Landon Dyer PD **MAKE**.

MS-DOS 2.x, 3.x, 4.x ***Just $99.95*** + $5 s/h Visa/MC
5 station LAN license $299.95 + $5 s/h, call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 **(919) 856-0475**

Number 288 on the Reader Service Card

DO IT YOURSELF!

XenoCopy-PC
PC-DOS program lets your PC
Read/Write/Format
over 350 formats **$79.95**

XenoFont
high quality text screen printouts
$49.95

+ $5.00 S/H + Tax if CA.
XenoSoft • (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Number 123 on the Reader Service Card

"QUICK & CLEAN" COBOL

The COBOL Report Activator (COBRA):

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and COPY members
- Produces audit trail reports
- Up to 5 output files per program
- Provides report options & site defined defaults
- Is available for Mainframe & PC

COBRA
Systems Software
David Olivadoti, P.O. Box 750817
Petaluma, CA 94975
(707) 763-5123

Number 281 on the Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities

COMPUTER BOOKS
Graphics, OS/2, Assembly

HARDWARE UPGRADES

(800) 423-3400
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Number 155 on the Reader Service Card

C SOURCE CODE

FOR ACCOUNTING BUSINESS AND MANUFACTURING MANAGEMENT

XLN offers K&R source code modules for general ledger, sales and accounts receivable, purchasing and accounts payable, inventory, bill of materials, MRP, CRP, PERT, maintenance management, personnel and payroll, and more.

XLN SOFTWARE
3680 Tuggle Road
Buford, GA 30518
800-543-5770
FAX: 404-859-0051
GA, Int'l or support: 404-945-8745

Number 237 on the Reader Service Card

PCHRT V3.00

HIGH RESOLUTION TIMER TOOLBOX

The definitive answer to execution profiling and embedded timer problems. 30 functions manage 100 software timers with microsecond accuracy. Self calibrating. Extensive and flexible timer reporting. Profiles selected ROM BIOS and MSDOS interrupts.

Supports TP 4 & 5, TC, MSC 5.x and QC 2.0. Libraries, manual, examples, and full source included.

$24.95 ppd USA, elsewhere add $4.00, Mich add 4%. VISA/MC

RYLE DESIGN
P.O. Box 22
Mt. Pleasant, MI 48804
(517) 773-0587

Number 210 on the Reader Service Card

D-ISAM

Industry standard file manager. Only $145 for DOS libraries, $595 for UNIX/DOS source code.

'W'

UNIX/DOS character windowing. $95 for DOS libraries, $295 for UNIX/DOS source code.

BYTE DESIGNS LTD.
#6-3511 Viking Way
Richmond, B.C.
V6V 1W1 CANADA
(800) 663-8547
(604) 278-5200

Number 219 on the Reader Service Card

MIDNIGHT ENGINEERING™

A new magazine for entrepreneurial hardware and software engineers that will encourage and challenge you to personally develop and market your own products.

- **PRACTICAL ARTICLES**
- **INSIGHTFUL INTERVIEWS**
- **DETAILED PRODUCT REVIEWS**

This is your chance to influence a brand-new magazine. PJ readers are invited to a **FREE** copy of the premiere issue of *Midnight Engineering* by calling or writing us with ideas.

DATArx
111 E. Drake Rd., Suite 7041
Fort Collins, CO 80525
303-223-2120

Number 272 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
282	Acucobol Inc.	16
30	The Austin Code Works	31
207	Automation Education	95
51	Az-Tech Software, Inc.	23
244	Block Island Technologies	53
3	Borland International	C4
288	Burton Systems Software	95
286	Business Computer Services	81
219	Byte Designs, Ltd.	95
*	CNS, Inc.	37
281	COBRA Systems Software	95
277	Covox Inc.	43
5	Dair Computer Systems	95
272	DATArx	95
229	Disk Software, Inc.	23
241	Emerald City Software	8
243	English Knowledge Systems	81
267	Falk Data Systems	63
274	Genesis Data Systems	7
261	Genus Microprogramming	6
*	Gimpel Software	79
285	Graphic Software Systems	26
122	Hauppauge Computer Works, Inc.	C3
154	HavenTree Software Limited	18
258	Helios Software	39
280	Helpful Programs	95
226	Ithaca Street Software, Inc.	49
211	KADAK Products Ltd.	26
155	Komputerwerk	95
63	Laboratory Microsystems Incorporated	63
44	Lahey Computer Systems Inc.	7
246	Logical Alternatives Incorporated	37
271	Lugaru Software Ltd.	19
212	Magna Carta Software	80
230	McSoft	91
143	Metagraphics Software Corporation	25
45	MetaWare Incorporated	1
232	Microsoft Corporation–Human Resources	9
104	Microway	15
121	MISOSYS, Inc.	51
238	MISOSYS, Inc., Hardware	30
266	Mix Software	81
94	MMC AD Systems	53
109	Nanosoft	87
160	Nu-Mega Technologies	3
283	NWP Intelligent Solutions	95
16	Opt-Tech Data Processing	47
254	OPUS Software	85
270	OTG Systems Inc.	30
*	Paradigm Systems	23
208	PC Tech	26
6	The Periscope Company	66
*	***PJ Code DiskScription***	8
*	***Programmer's Journal*** Mail Order Store	88
228	ProTech Marketing, Inc.	94
97	Qualstar Corporation	23
252	Quantum Software	60
147	Quarterdeck Office Systems	C2
251	Radix Systems	70
222	Rainbow Technologies	11
20	Rational Systems, Inc.	77
198	The Research Group	71
210	Ryle Design	95
269	Sequiter Software Inc.	68
165	SLR Systems	22
33	Soft Advances	14
275	Software Development Technologies	53
98	The Software Engineering Store	89
221	Software Security	5
10	South Mountain Software	52
158	Summit Information Systems, Inc.	38
289	TechnoJock Software Inc.	74
264	Techsel Corporation	18
166	The Tool Makers	39
276	Tri-Technology Systems Inc.	43
25	TurboPower Software	17
259	Turbo User Group	37
290	Ultra-X	60
125	V Communications, Inc.	58
100	Western Wares	63
73	The Whitewater Group	34
123	Xenosoft	95
237	XLN Software	95

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers with an * prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors.

COMING ISSUES

September/October
Object-Oriented Programming

Object-Oriented Development Tools *by Vaughn Vernon*
Better C++ Code Generation *by John Dlugosz*

PJ Forum:
The Schism between RISC and CISC
by Hal Hardenbergh

On Graphics:
360x480 VGA Resolution and Panning
by Michael Abrash

November/December
User Interface Design

Advertise in *Programmer's Journal*
September/October 1989 Volume 7.5

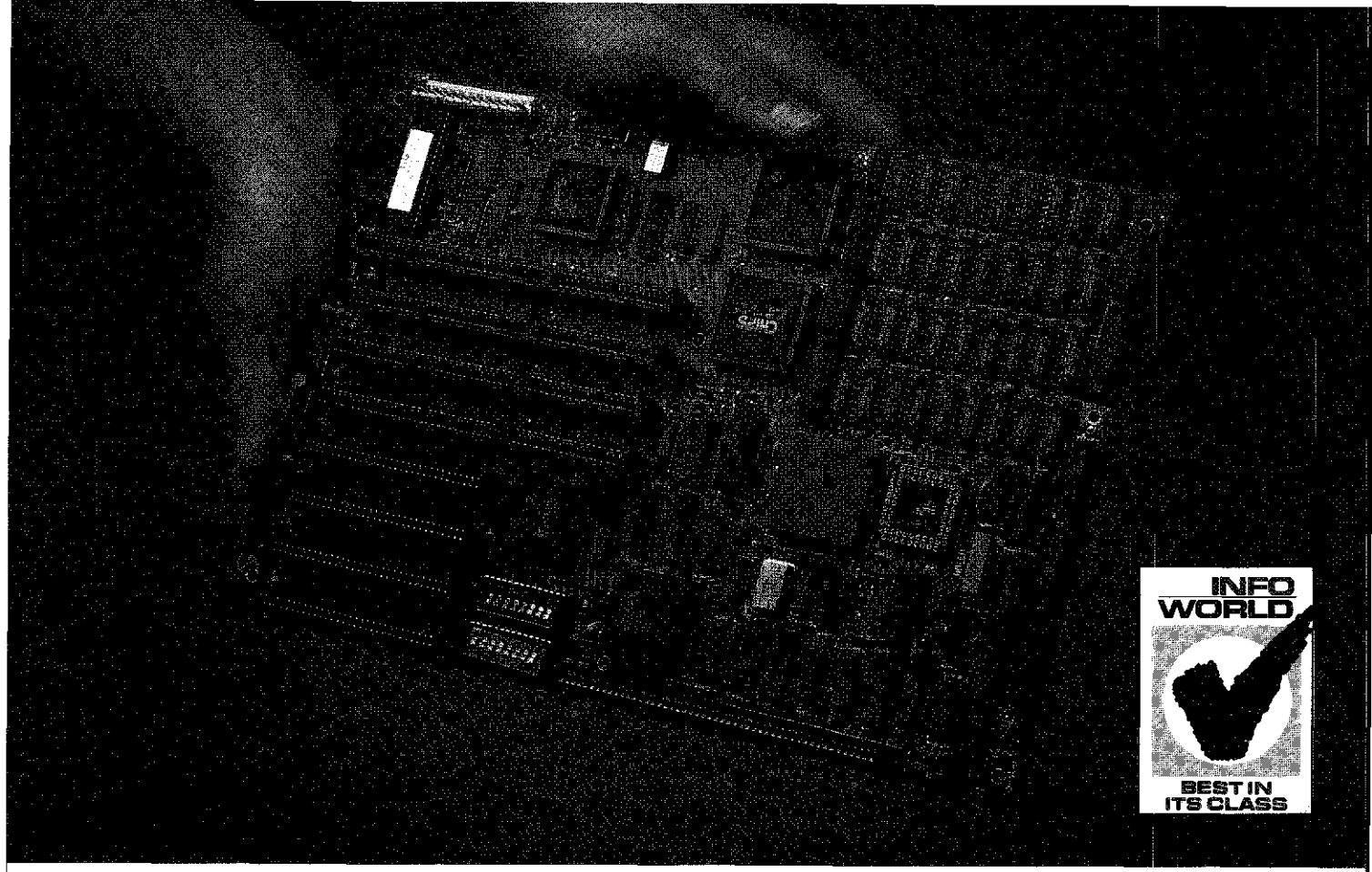
Space Reservation Deadline:
July 7, 1989

Materials Due:
July 21, 1989

Please call *Programmer's Journal* to reserve your space.

(503) 747-0800

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



IT Hz SO GOOD!

INTRODUCING HAUPPAUGE'S 33MHz SYSTEM BOARDS

Do you need the highest multiuser, network or CAD performance? With a 386 MotherBoard built to handle 33MHz operation, you can configure the fastest computer on the block!

The UNIX Engine. Great for VARS, Systems Integrators and UNIX OEMs, our 386 MotherBoards run SCO Xenix, Interactive 386/ix and AT&T's UNIX System V. With its PC/AT compatible I/O system, the Hauppauge 386 MotherBoard/33MHz accepts the latest in high performance disk control, graphics, and network I/O cards — just to name a few.

Elephant Memory. With up to 64MBytes of high speed 32-bit memory, your UNIX systems can accommodate many users who demand top performance.

Network Savvy. Ethernet, token ring, TCP/IP are all compatible with our 386 MotherBoards.

High Performance Math Capability. For your CAD and graphics applications, the 33MHz 386 MotherBoard supports the super high speed 387-33 and Weitek Abacus math coprocessors.

Technical Features ▪ 16, 20, 25, and 33MHz 80386 ▪ from 1 to 64 Mbytes of high speed 32-bit 4-way interleaved RAM ▪ PC/AT compatible BIOS and I/O system for support of OS/2 and UNIX ▪ 64K of 20 nsec static RAM cache on 25MHz and 33MHz boards ▪ One 32-bit expansion slot for up to 64 Megabytes of high speed memory.

For more about our products, mail us the coupon below.

Yes, send me your product information!

Name_____

Company_____

Address_____

City, State, Country_____

Telephone_____

Mail coupon to West Germany or the U.S.A.

Hauppauge Computer Works, GmbH
Hansaallee 201
4000 Dusseldorf 11, West Germany
Tel: 0211-594320
Fax: 0211-593908

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788, U.S.A.
Tel: 01-516-434-1600
Fax: 01-516-434-3198

If your computer feels slow, we know where it hertz. Call for a fast cure: In the US: 516-434-1600. In Europe: 49-211-594320.

Hauppauge Computer Works
Your high performance 386 Supplier

Hauppauge!

Trademarks: IBM PC, XT, AT, PS2 and OS/2: IBM. Intel 386: Intel Corp.; Windows/386: Microsoft Corp. DESQview: Quarterdeck

Number **122** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BORLAND INTRODUCES TURBO PASCAL 5.5 WITH OBJECTS

Be Objective.

Turbo Pascal,® the world-standard Pascal compiler, adds Object-Oriented Programming with our new version 5.5. We combined the simplicity of Apple's Object Pascal language with the power and efficiency of C++ to create Turbo Pascal 5.5, the object-oriented programming language for the rest of us.

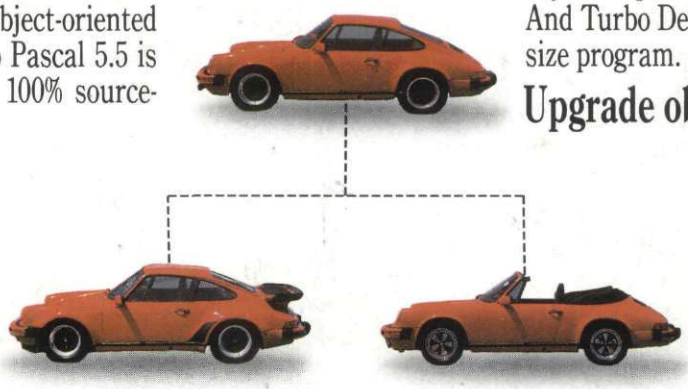
It's easy to extend yourself

If you're already programming with Turbo Pascal, it's easy to extend yourself from structured programming to object-oriented programming. And, Turbo Pascal 5.5 is the *only* compiler that is 100% source-code compatible with your existing Turbo Pascal 4.0 and 5.0 programs.

A fast object lesson

Object-oriented application programs more closely model the way you think. Objects contain both data and code. As in a spreadsheet cell, the value and the formula are together. Objects can *inherit* properties from other objects. For example, a Porsche Carrera inherits most attributes from the base model 911, but it also sports a whale tail.

Turbo Pascal 5.5's object-oriented extensions give you code that's easier to change, extend, and support.



Inheritance provides powerful modeling capabilities by allowing objects to inherit attributes from other objects.

Turbo Pascal 5.5 Professional with Turbo Debugger® and Turbo Assembler®

The award-winning Turbo Debugger now includes an object inspector and hierarchy browser. And Turbo Debugger can debug any size program.

Upgrade objectively

Pascal owners: Upgrading from Turbo Pascal 5.0 to 5.5 is only $34.95 plus $5 shipping and handling ($75 plus shipping and handling for owners of Turbo Pascal 4.0 or earlier). And upgrading from Turbo Pascal 5.0 and earlier to Turbo Pascal 5.5 Professional is only $99.95 plus $10 shipping and handling. To order, CALL (800) 331-0877.

Turbo Pascal 5.5 Features

- Inheritance
- Static & dynamic objects
- Constructors & Destructors
- Object constants
- Compiles @ > 34,000 lines/minute
- New integrated environment tutorial
- Hypertext Help with copy and paste
- Enhanced smart linker & overlay manager
- Support for 8087/80287/80387
- Integrated source-level debugging

BORLAND

Code: MA13

Mail upgrade orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95066-0001. For orders outside the U.S., call (408) 438-5300.

Turbo Pascal, Turbo Debugger, and *Turbo Assembler* are registered trademarks of Borland International. Copyright ©1989, Borland International, Inc. All right reserved. BI 1324

Number 3 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED